The Org Manual

www.gnu.org/software/emacs/manual/html_mono/org.html

This manual is for Org version 9.7.

1 Introduction

1.1 Summary

Org Mode is an authoring tool and a TODO lists manager for GNU Emacs. It relies on a lightweight plain-text markup language used in files with the '`.org`' extension.

Authoring Org files is best supported by Emacs, but you can view, understand, and change them with any text editor.

As an authoring tool, Org helps you write structured documents and provides exporting facilities. Org files can also be used for literate programming and reproducible research. As a TODO lists manager, Org helps you organize your tasks in a flexible way, from daily needs to detailed project-planning, allowing logging, multiple views on your tasks, exporting your agendas, etc.

Org mode is implemented on top of Outline mode, which makes it possible to keep the content of large files well structured. Visibility cycling and structure editing help to work with the tree. Tables are easily created with a built-in table editor. Plain text URL-like links connect to websites, emails, Usenet messages, BBDB entries, and any files related to the projects.

Org develops organizational tasks around notes files that contain lists or information about projects as plain text. Project planning and task management make use of metadata which is part of an outline node. Based on this data, specific entries can be extracted in queries and create dynamic *agenda views* that also integrate the Emacs calendar and diary. Org can be used to implement many different project planning schemes, such as David Allen's GTD system.

Org files can serve as a single source authoring system with export to many different formats such as HTML, LaTeX, Open Document, and Markdown. New export backends can be derived from existing ones, or defined from scratch.

Org files can include source code blocks, which makes Org uniquely suited for authoring technical documents with code examples. Org source code blocks are fully functional; they can be evaluated in place and their results can be captured in the file. This makes it possible to create a single file reproducible research compendium.

Org keeps simple things simple. When first fired up, it should feel like a straightforward, easy to use outliner. Complexity is not imposed, but a large amount of functionality is available when needed. Org is a toolbox. Many users actually run only a—very personal—fraction of Org's capabilities, and know that there is more whenever they need it.

All of this is achieved with strictly plain text files, the most portable and future-proof file format. Org runs in Emacs. Emacs is one of the most widely ported programs, so that Org mode is available on every major platform.

There is a website for Org which provides links to the newest version of Org, as well as additional information, frequently asked questions (FAQ), links to tutorials, etc. This page is located at https://orgmode.org.

An earlier version (7.3) of this manual was available as a paperback book from the Network Theory Ltd. publishing company, closed in 2009.

Next: Activation, Previous: Summary, Up: Introduction [Contents][Index]

1.2 Installation

Org is included in distributions of GNU Emacs, you probably do not need to install it. Most users will simply activate Org and begin exploring its features.

If, for one reason or another, you want to install Org on top of this pre-packaged version, you can use the Emacs package system or clone Org's git repository. We **strongly recommend** sticking to a single installation method.

When installing Org on top of the pre-packaged version, please note that Org stable versions are meant to be fully compatible with the last three stable versions of Emacs but not with older Emacsen.

Some Org components also depend on third-party packages available through package archives. Org is only guaranteed to be compatible with the latest stable versions of these third-party packages.

Using Emacs packaging system

Recent Emacs distributions include a packaging system which lets you install Elisp libraries. You can install Org from the "package menu", with `M-x list-packages`. See (emacs)Package Menu.

> **Important:** You need to do this in a session where no '`.org`' file has been visited, i.e., where no Org built-in function have been loaded. Otherwise autoload Org functions will mess up the installation.

To avoid interference with built-in Org mode, you can use command line:

```
emacs -Q -batch -eval "(progn (require 'package) (package-initialize) (package-refresh-contents) (package-upgrade 'org))"
```

This approach has the advantage of isolating the upgrade process from a running Emacs session, ensuring that version conflicts can not arise.

Using Org's git repository

You can clone Org's repository and install Org like this:

```
$ cd ~/src/
$ git clone https://git.savannah.gnu.org/git/emacs/org-mode.git
$ cd org-mode/
$ make autoloads
```

Note that in this case, '`make autoloads`' is mandatory: it defines Org's version in '`org-version.el`' and Org's autoloads in '`org-loaddefs.el`'.

Make sure you set the load path correctly in your Emacs init file:

```
(add-to-list 'load-path "~/src/org-mode/lisp")
```

You can also compile with '`make`', generate the documentation with '`make doc`', create a local configuration with '`make config`' and install Org with '`make install`'. Please run '`make help`' to get the list of compilation/installation options.

For more detailed explanations on Org's build system, please check the Org Build System page on Worg.

Installing Org's contributed packages

Org's repository used to contain '`contrib/`' directory for add-ons contributed by others. As of Org 9.5, the directory has been moved to the dedicated org-contrib repository, which you can install separately as a package from NonGNU ELPA.

There are enough valuable packages maintained outside of the Org repository. Worg has a list of org-contrib and external packages, certainly it is not exhaustive.

1.3 Activation

Org mode buffers need Font Lock to be turned on: this is the default in Emacs[1].

There are compatibility issues between Org mode and some other Elisp packages (see Packages that conflict with Org mode). Please take the time to check the list.

For a better experience, the three Org commands `org-store-link`, `org-capture` and `org-agenda` ought to be accessible anywhere in Emacs, not just in Org buffers. To that effect, you need to bind them to globally available keys, like the ones reserved for users (see (elisp)Key Binding Conventions). Here are suggested bindings, please modify the keys to your own liking in your (emacs)personal init file.

```
(global-set-key (kbd "C-c l") #'org-store-link)
(global-set-key (kbd "C-c a") #'org-agenda)
(global-set-key (kbd "C-c c") #'org-capture)
```

Files with the '`.org`' extension use Org mode by default. To turn on Org mode in a file that does not have the extension '`.org`', make the first line of a file look like this:

```
MY PROJECTS    -*- mode: org; -*-
```

which selects Org mode for this buffer no matter what the file's name is. See also the variable `org-insert-mode-line-in-empty-file`.

Many commands in Org work on the region if the region is *active*. To make use of this, you need to have Transient Mark mode turned on, which is the default. If you do not like it, you can create an active region by using the mouse to select a region, or pressing `C-SPC` twice before moving point.

Next: Typesetting Conventions Used in this Manual, Previous: Activation, Up: Introduction [Contents][Index]

1.4 Feedback

If you find problems with Org, or if you have questions, remarks, or ideas about it, please send an email to the Org mailing list mailto:emacs-orgmode@gnu.org. You can subscribe to the list from this web page. If you are not a member of the mailing list, your mail will be passed to the list after a moderator has approved it[2]. We ask you to read and respect the GNU Kind Communications Guidelines when sending messages on this mailing list. Please allow up to one month for the response and followup if no response is received on the bug report.

For bug reports, please first try to reproduce the bug with the latest version of Org available—if you are running an outdated version, it is quite possible that the bug has been fixed already. If the bug persists, prepare a report and provide as much information as possible, including the version information of Emacs (`M-x emacs-version`) and Org (`M-x org-version`), as well as the Org related setup in the Emacs init file. The easiest way to do this is to use the command

```
M-x org-submit-bug-report <RET>
```

which puts all this information into an Emacs mail buffer so that you only need to add your description. If you are not sending the Email from within Emacs, please copy and paste the content into your Email program.

Sometimes you might face a problem due to an error in your Emacs or Org mode setup. Before reporting a bug, it is very helpful to start Emacs with minimal customizations and reproduce the problem. Doing so often helps you determine if the problem is with your customization or with Org mode itself. You can start a typical minimal session with a command like the example below.

```
$ emacs -Q -l /path/to/minimal-org.el
```

However if you are using Org mode as distributed with Emacs, a minimal setup is not necessary. In that case it is sufficient to start Emacs as '`emacs -Q`'. The '`minimal-org.el`' setup file can have contents as shown below.

```
;;; Minimal setup to load latest `org-mode'.

;; Activate debugging.
(setq debug-on-error t
      debug-on-signal nil
      debug-on-quit nil)

;; Add latest Org mode to load path.
(add-to-list 'load-path (expand-file-name "/path/to/org-mode/lisp"))
```

If you are using Org mode version from Git repository, you can start minimal session using make.

```
# Bare Emacs
make repro
# or pass extra arguments
make repro REPRO_ARGS="-l /path/to/minimal/config.el /tmp/bug.org"
```

If an error occurs, a “backtrace” can be very useful—see below on how to create one. Often a small example file helps, along with clear information about:

1. What exactly did you do?
2. What did you expect to happen?
3. What happened instead?

If you experience degraded performance, you can record a “profile” and share it on the Org mailing list. See below for the instructions how to record a useful profile.

Thank you for helping to improve this program.

How to create a useful backtrace

If working with Org produces an error with a message you do not understand, you may have hit a bug. The best way to report this is by providing, in addition to what was mentioned above, a backtrace. This is information from the built-in debugger about where and how the error occurred. Here is how to produce a useful backtrace:

1. Reload uncompiled versions of all Org mode Lisp files. The backtrace contains much more information if it is produced with uncompiled code. To do this, use

   ```
   C-u M-x org-reload <RET>
   ```

   or, from the menu: Org → Refresh/Reload → Reload Org uncompiled.

2. Then, activate the debugger:

   ```
   M-x toggle-debug-on-error <RET>
   ```

   or, from the menu: Options → Enter Debugger on Error.

3. Do whatever you have to do to hit the error. Do not forget to document the steps you take.
4. When you hit the error, a ‘`*Backtrace*`’ buffer appears on the screen. Save this buffer to a file—for example using `C-x C-w`—and attach it to your bug report.

How to profile Org performance

Sometimes, Org is becoming slow for no apparent reason. Such slowdown is often caused by interaction between third-party packages and Org mode. However, identifying the root cause is not always straightforward.

Emacs is able to record performance statistics, which can then be used to find out which functions are taking most of the time to execute. To record the statistics, one can use so-called profiler. To use the Emacs profiler, we recommend the following steps:

1. Make sure that no profiler is currently active:

   ```
   M-x profiler-stop <RET>
   ```

2. Start a new CPU profiler session:

   ```
   M-x profiler-start <RET> cpu <RET>
   ```

3. Use Emacs as usual, performing the actions that are deemed slow.
4. Display and examine the recorded performance statistics:

   ```
   M-x profiler-report <RET>
   ```

   This command will display a summary of the commands and functions that have been executed between `profiler-start` and `profiler-report` invocations, with command taking most of the time displayed on top.

   '<TAB>' key can be used to fold and unfold lines in the profiler buffer. The child items revealed upon unfolding are the functions and commands called by the unfolded parent.

   The root causes are often buried deep inside sub-children items in the profiler. You can press 'B' (`profiler-report-render-reversed-calltree`) to quickly reveal the actual function/command that takes most of the time to run.

   Pressing 'C' `profiler-report-render-calltree` will recover the original view.

5. If you need further help, you can share the statistics data.
   Just save the data by issuing

   ```
   M-x profiler-report-write-profile <RET>
   /path/to/profile-file-to-be-saved <RET>
   ```

   Then, you can attached the saved file to your email to the Org mailing list, alongside with details about what you did to trigger the slowdown.

   Note that the saved statistics will only contain the function names and how long their execution takes. No private data will be recorded.

Previous: Feedback, Up: Introduction [Contents][Index]

1.5 Typesetting Conventions Used in this Manual

TODO keywords, tags, properties, etc.

Org uses various syntactical elements: TODO keywords, tags, property names, keywords, blocks, etc. In this manual we use the following conventions:

'TODO'
'WAITING'
TODO keywords are written with all capitals, even if they are user-defined.

'boss'
'ARCHIVE'
Tags are case-sensitive. User-defined tags are usually written in lowercase; built-in tags with special meaning are written as they should appear in the document, usually with all capitals.

'Release'

‘PRIORITY’
User-defined properties are capitalized; built-in properties with special meaning are written with all capitals.

‘TITLE’
‘BEGIN’ … ‘END’
Keywords and blocks are written in uppercase to enhance their readability, but you can use lowercase in your Org files.

Key bindings and commands

The manual lists both the keys and the corresponding commands for accessing a functionality. Org mode often uses the same key for different functions, depending on context. The command that is bound to such keys has a generic name, like `org-metaright`. In the manual we will, wherever possible, give the function that is internally called by the generic command. For example, in the chapter on document structure, `M-RIGHT` will be listed to call `org-do-demote`, while in the chapter on tables, it will be listed to call `org-table-move-column-right`.

Next: Tables, Previous: Introduction, Up: The Org Manual [Contents][Index]

2 Document Structure

Org is an outliner. Outlines allow a document to be organized in a hierarchical structure, which, least for me, is the best representation of notes and thoughts. An overview of this structure is achieved by folding, i.e., hiding large parts of the document to show only the general document structure and the parts currently being worked on. Org greatly simplifies the use of outlines by compressing the entire show and hide functionalities into a single command, `org-cycle`, which is bound to the `TAB` key.

Next: Visibility Cycling, Up: Document Structure [Contents][Index]

2.1 Headlines

Headlines define the structure of an outline tree. Org headlines start on the left margin[3] with one or more stars followed by a space. For example:

```
* Top level headline
** Second level
*** Third level
    some text
*** Third level
    more text
* Another top level headline
```

The name defined in `org-footnote-section` is reserved. Do not use it as a title for your own headings.

Some people find the many stars too noisy and would prefer an outline that has whitespace followed by a single star as headline starters. This can be achieved using an Org Indent minor mode. See A Cleaner Outline View for more information.

Headlines are not numbered. However, you may want to dynamically number some, or all, of them. See Dynamic Headline Numbering.

An empty line after the end of a subtree is considered part of it and is hidden when the subtree is folded. However, if you leave at least two empty lines, one empty line remains visible after folding the subtree, in order to structure the collapsed view. See the variable `org-cycle-separator-lines` to modify this behavior.

2.2 Visibility Cycling

2.2.1 Global and local cycling

Outlines make it possible to hide parts of the text in the buffer. Org uses just two commands, bound to `TAB` and `S-TAB` to change the visibility in the buffer.

`TAB` (**`org-cycle`**) ¶
Subtree cycling: Rotate current subtree among the states

```
,-> FOLDED -> CHILDREN -> SUBTREE --.
'-----------------------------------'
```

Point must be on a headline for this to work[4].

`S-TAB` (**`org-global-cycle`**) ¶
`C-u TAB`
Global cycling: Rotate the entire buffer among the states

```
,-> OVERVIEW -> CONTENTS -> SHOW ALL --.
'--------------------------------------'
```

When `S-TAB` is called with a numeric prefix argument *N*, view contents only up to headlines of level *N*.

Note that inside tables (see Tables), `S-TAB` jumps to the previous field instead.

You can run global cycling using `TAB` only if point is at the very beginning of the buffer, but not on a headline, and `org-cycle-global-at-bob` is set to a non-`nil` value.

`C-u C-u TAB` (**`org-set-startup-visibility`**) ¶
Switch back to the startup visibility of the buffer (see Initial visibility).

`C-u C-u C-u TAB` (**`org-show-all`**) ¶
Show all, including drawers.

`C-c C-r` (**`org-reveal`**) ¶
Reveal context around point, showing the current entry, the following heading and the hierarchy above. It is useful for working near a location that has been exposed by a sparse tree command (see Sparse Trees) or an agenda command (see Commands in the Agenda Buffer). With a prefix argument, show, on each level, all sibling headings. With a double prefix argument, also show the entire subtree of the parent.

`C-c C-k` (**`org-show-branches`**) ¶
Expose all the headings of the subtree, but not their bodies.

`C-c TAB` (`org-show-children`) ¶
Expose all direct children of the subtree. With a numeric prefix argument *N*, expose all children down to level *N*.

`C-c C-x b` (`org-tree-to-indirect-buffer`) ¶
Show the current subtree in an indirect buffer[5]. With a numeric prefix argument *N*, go up to level *N* and then take that tree. If *N* is negative then go up that many levels. With a `C-u` prefix, do not remove the previously used indirect buffer.

`C-c C-x v` (`org-copy-visible`) ¶
Copy the *visible* text in the region into the kill ring.

2.2.2 Initial visibility

When Emacs first visits an Org file, the global state is set to `showeverything`, i.e., all file content is visible[6]. This can be configured through the variable `org-startup-folded`, or on a per-file basis by adding one of the following lines anywhere in the buffer:

```
#+STARTUP: overview
#+STARTUP: content
#+STARTUP: showall
#+STARTUP: show2levels
#+STARTUP: show3levels
#+STARTUP: show4levels
#+STARTUP: show5levels
#+STARTUP: showeverything
```

Furthermore, any entries with a ‘`VISIBILITY`’ property (see Properties and Columns) get their visibility adapted accordingly. Allowed values for this property are ‘`folded`’, ‘`children`’, ‘`content`’, and ‘`all`’.

`C-u C-u TAB` (`org-set-startup-visibility`) ¶
Switch back to the startup visibility of the buffer, i.e., whatever is requested by startup options and ‘`VISIBILITY`’ properties in individual entries.

2.2.3 Catching invisible edits

Sometimes you may inadvertently edit an invisible part of the buffer and be confused on what has been edited and how to undo the mistake. By default, Org prevents such edits for a limited set of user commands. Users can control which commands are affected by customizing `org-fold-catch-invisible-edits-commands`.

The strategy used to decide if a given edit is dangerous is controlled by `org-fold-catch-invisible-edits`. See the docstring of this option on the available strategies. Set the option to `nil` to disable catching invisible edits completely.

Next: Structure Editing, Previous: Visibility Cycling, Up: Document Structure [Contents][Index]

2.3 Motion

The following commands jump to other headlines in the buffer.

`C-c C-n` (`org-next-visible-heading`) ¶

Next heading.

`C-c C-p` (`org-previous-visible-heading`) ¶
Previous heading.

`C-c C-f` (`org-forward-heading-same-level`) ¶
Next heading same level.

`C-c C-b` (`org-backward-heading-same-level`) ¶
Previous heading same level.

`C-c C-u` (`outline-up-heading`) ¶
Backward to higher level heading.

`C-c C-j` (`org-goto`) ¶
Jump to a different place without changing the current outline visibility. Shows the document structure in a temporary buffer, where you can use the following keys to find your destination:

`TAB`	Cycle visibility.
`DOWN` / `UP`	Next/previous visible headline.
`RET`	Select this location.
`/`	Do a Sparse-tree search

The following keys work if you turn off `org-goto-auto-isearch`

`n` / `p`	Next/previous visible headline.
`f` / `b`	Next/previous headline same level.
`u`	One level up.
`0` … `9`	Digit argument.
`q`	Quit.

See also the variable `org-goto-interface`.

Next: Sparse Trees, Previous: Motion, Up: Document Structure [Contents][Index]

2.4 Structure Editing

`M-RET` (`org-meta-return`) ¶
Insert a new heading, item or row.

If the command is used at the *beginning* of a line, and if there is a heading or a plain list item (see Plain Lists) at point, the new heading/item is created *before* the current line. When used at the beginning of a regular line of text, turn that line into a heading.

When this command is used in the middle of a line, the line is split and the rest of the line becomes the new item or headline. If you do not want the line to be split, customize `org-M-RET-may-split-line`.

Calling the command with a `C-u` prefix unconditionally inserts a new heading at the end of the current subtree, thus preserving its contents. With a double `C-u C-u` prefix, the new heading is created at the end of the parent subtree instead.

`C-RET` (**`org-insert-heading-respect-content`**) ¶
Insert a new heading at the end of the current subtree.

`M-S-RET` (**`org-insert-todo-heading`**) ¶
Insert new TODO entry with same level as current heading. See also the variable `org-treat-insert-todo-heading-as-state-change`.

`C-S-RET` (**`org-insert-todo-heading-respect-content`**) ¶
Insert new TODO entry with same level as current heading. Like `C-RET`, the new headline is inserted after the current subtree.

`TAB` (**`org-cycle`**) ¶
In a new entry with no text yet, the first `TAB` demotes the entry to become a child of the previous one. The next `TAB` makes it a parent, and so on, all the way to top level. Yet another `TAB`, and you are back to the initial level.

`M-LEFT` (**`org-do-promote`**) ¶
`M-RIGHT` (**`org-do-demote`**)
Promote or demote current heading by one level.

When there is an active region—i.e., when Transient Mark mode is active—promotion and demotion work on all headlines in the region. To select a region of headlines, it is best to place both point and mark at the beginning of a line, mark at the beginning of the first headline, and point at the line just after the last headline to change.

`M-S-LEFT` (**`org-promote-subtree`**) ¶
Promote the current subtree by one level.

`M-S-RIGHT` (**`org-demote-subtree`**) ¶
Demote the current subtree by one level.

`M-UP` (**`org-move-subtree-up`**) ¶
Move subtree up, i.e., swap with previous subtree of same level.

`M-DOWN` (**`org-move-subtree-down`**) ¶
Move subtree down, i.e., swap with next subtree of same level.

`C-c @` (**`org-mark-subtree`**) ¶
Mark the subtree at point. Hitting repeatedly marks subsequent subtrees of the same level as the marked subtree.

`C-c C-x C-w` (**`org-cut-subtree`**) ¶

Kill subtree, i.e., remove it from buffer but save in kill ring. With a numeric prefix argument N, kill N sequential subtrees.

`C-c C-x M-w` (`org-copy-subtree`) ¶
Copy subtree to kill ring. With a numeric prefix argument N, copy the N sequential subtrees.

`C-c C-x C-y` (`org-paste-subtree`) ¶
Yank subtree from kill ring. This does modify the level of the subtree to make sure the tree fits in nicely at the yank position. The yank level can also be specified with a numeric prefix argument, or by yanking after a headline marker like '`****`'. With `C-u` prefix, force inserting as a sibling. With `C-u C-u` prefix argument, force inserting as a child.

`C-y` (`org-yank`) ¶
Depending on the variables `org-yank-adjusted-subtrees` and `org-yank-folded-subtrees`, Org's internal `yank` command pastes subtrees folded and in a clever way, using the same command as `C-c C-x C-y`. With the default settings, no level adjustment takes place, but the yanked tree is folded unless doing so would swallow text previously visible. Any prefix argument to this command forces a normal `yank` to be executed, with the prefix passed along. A good way to force a normal yank is `C-u C-y`. If you use `yank-pop` after a yank, it yanks previous kill items plainly, without adjustment and folding.

`C-c C-x c` (`org-clone-subtree-with-time-shift`) ¶
Clone a subtree by making a number of sibling copies of it. You are prompted for the number of copies to make, and you can also specify if any timestamps in the entry should be shifted. This can be useful, for example, to create a number of tasks related to a series of lectures to prepare. For more details, see the docstring of the command `org-clone-subtree-with-time-shift`.

`C-c C-w` (`org-refile`) ¶
Refile entry or region to a different location. See Refile and Copy.

`C-c ^` (`org-sort`) ¶
Sort same-level entries. When there is an active region, all entries in the region are sorted. Otherwise the children of the current headline are sorted. The command prompts for the sorting method, which can be alphabetically, numerically, by time—first timestamp with active preferred, creation time, scheduled time, deadline time—by priority, by TODO keyword—in the sequence the keywords have been defined in the setup—or by the value of a property. Reverse sorting is possible as well. You can also supply your own function to extract the sorting key. With a `C-u` prefix, sorting is case-sensitive.

`C-x n s` (`org-narrow-to-subtree`) ¶
Narrow buffer to current subtree.

`C-x n b` (`org-narrow-to-block`) ¶
Narrow buffer to current block.

`C-x n w` (`widen`) ¶
Widen buffer to remove narrowing.

`C-c *` (`org-toggle-heading`) ¶

Turn a normal line or plain list item into a headline—so that it becomes a subheading at its location. Also turn a headline into a normal line by removing the stars. If there is an active region, turn all lines in the region into headlines. If the first line in the region was an item, turn only the item lines into headlines. Finally, if the first line is a headline, remove the stars from all headlines in the region.

Note that when point is inside a table (see Tables), the Meta-Cursor keys have different functionality.

2.5 Sparse Trees

An important feature of Org mode is the ability to construct *sparse trees* for selected information in an outline tree, so that the entire document is folded as much as possible, but the selected information is made visible along with the headline structure above it[7]. Just try it out and you will see immediately how it works.

Org mode contains several commands creating such trees, all these commands can be accessed through a dispatcher:

`C-c /` (`org-sparse-tree`) ¶
This prompts for an extra key to select a sparse-tree creating command.

`C-c / r` or **`C-c / /`** (`org-occur`) ¶
Prompts for a regexp (see Regular Expressions) and shows a sparse tree with all matches. If the match is in a headline, the headline is made visible. If the match is in the body of an entry, headline and body are made visible. In order to provide minimal context, also the full hierarchy of headlines above the match is shown, as well as the headline following the match. Each match is also highlighted; the highlights disappear when the buffer is changed by an editing command, or by pressing `C-c C-c`[8]. When called with a `C-u` prefix argument, previous highlights are kept, so several calls to this command can be stacked.

`M-g n` or **`M-g M-n`** (`next-error`) ¶
Jump to the next sparse tree match in this buffer.

`M-g p` or **`M-g M-p`** (`previous-error`) ¶
Jump to the previous sparse tree match in this buffer.

For frequently used sparse trees of specific search strings, you can use the variable `org-agenda-custom-commands` to define fast keyboard access to specific sparse trees. These commands will then be accessible through the agenda dispatcher (see The Agenda Dispatcher). For example:

```
(setq org-agenda-custom-commands
      '(("f" occur-tree "FIXME")))
```

defines the key `f` as a shortcut for creating a sparse tree matching the string ‘FIXME’.

The other sparse tree commands select headings based on TODO keywords, tags, or properties and are discussed later in this manual.

To print a sparse tree, you can use the Emacs command `ps-print-buffer-with-faces` which does not print invisible parts of the document. Or you can use the command `C-c C-e C-v` to export only the visible part of the document and print the resulting file.

2.6 Plain Lists

Within an entry of the outline tree, hand-formatted lists can provide additional structure. They also provide a way to create lists of checkboxes (see Checkboxes). Org supports editing such lists, and every exporter (see Exporting) can parse and format them.

Org knows ordered lists, unordered lists, and description lists.

- *Unordered* list items start with '`-`', '`+`', or '`*`'[9] as bullets.
- *Ordered* list items start with a numeral followed by either a period or a right parenthesis[10], such as '`1.`' or '`1)`'[11] If you want a list to start with a different value—e.g., 20—start the text of the item with '`[@20]`'[12]. Those constructs can be used in any item of the list in order to enforce a particular numbering.
- *Description* list items are unordered list items, and contain the separator '`::`' to distinguish the description *term* from the description.

Items belonging to the same list must have the same indentation on the first line. In particular, if an ordered list reaches number '`10.`', then the 2-digit numbers must be written left-aligned with the other numbers in the list. An item ends before the next line that is less or equally indented than its bullet/number.

A list ends whenever every item has ended, which means before any line less or equally indented than items at top level. It also ends before two blank lines. In that case, all items are closed. Here is an example:

```
* Lord of the Rings
My favorite scenes are (in this order)
1. The attack of the Rohirrim
2. Eowyn's fight with the witch king
   + this was already my favorite scene in the book
   + I really like Miranda Otto.
3. Peter Jackson being shot by Legolas
   - on DVD only
   He makes a really funny face when it happens.
8. [@8] <favorite scenes 4 to 8 are skipped for brevity>
But in the end, no individual scenes matter but the film as a whole.
Important actors in this film are:
- Elijah Wood :: He plays Frodo
- Sean Astin :: He plays Sam, Frodo's friend.  I still remember him
     very well from his role as Mikey Walsh in /The Goonies/.
```

Org supports these lists by tuning filling and wrapping commands to deal with them correctly, and by exporting them properly (see Exporting). Since indentation is what governs the structure of these lists, many structural constructs like '`#+BEGIN_`' blocks can be indented to signal that they belong to a particular item.

If you find that using a different bullet for a sub-list—than that used for the current list-level—improves readability, customize the variable `org-list-demote-modify-bullet`. To get a greater difference of indentation between items and theirs sub-items, customize `org-list-indent-`

`offset`.

The following commands act on items when point is in the first line of an item—the line with the bullet or number. Some of them imply the application of automatic rules to keep list structure intact. If some of these actions get in your way, configure `org-list-automatic-rules` to disable them individually.

`TAB` (`org-cycle`) ¶
Items can be folded just like headline levels. Normally this works only if point is on a plain list item. For more details, see the variable `org-cycle-include-plain-lists`. If this variable is set to `integrate`, plain list items are treated like low-level headlines. The level of an item is then given by the indentation of the bullet/number. Items are always subordinate to real headlines, however; the hierarchies remain completely separated. In a new item with no text yet, the first `TAB` demotes the item to become a child of the previous one. Subsequent `TAB`s move the item to meaningful levels in the list and eventually get it back to its initial position.

`M-RET` (`org-insert-heading`) ¶
Insert new item at current level. With a prefix argument, force a new heading (see Structure Editing). If this command is used in the middle of an item, that item is *split* in two, and the second part becomes the new item[13]. If this command is executed *before item's body*, the new item is created *before* the current one.

`M-S-RET` ¶
Insert a new item with a checkbox (see Checkboxes).

`S-UP` ¶
`S-DOWN`
Jump to the previous/next item in the current list, but only if `org-support-shift-select` is off[14]. If not, you can still use paragraph jumping commands like `C-UP` and `C-DOWN` to quite similar effect.

`M-UP` ¶
`M-DOWN`
Move the item including subitems up/down[15], i.e., swap with previous/next item of same indentation. If the list is ordered, renumbering is automatic.

`M-LEFT` ¶
`M-RIGHT`
Decrease/increase the indentation of an item, leaving children alone.

`M-S-LEFT` ¶
`M-S-RIGHT`
Decrease/increase the indentation of the item, including subitems. Initially, the item tree is selected based on current indentation. When these commands are executed several times in direct succession, the initially selected region is used, even if the new indentation would imply a different hierarchy. To use the new hierarchy, break the command chain by moving point.

As a special case, using this command on the very first item of a list moves the whole list. This behavior can be disabled by configuring `org-list-automatic-rules`. The global indentation of a list has no influence on the text *after* the list.

`C-c C-c` ¶
If there is a checkbox (see Checkboxes) in the item line, toggle the state of the checkbox. In any case, verify bullets and indentation consistency in the whole list.

`C-c -` ¶
Cycle the entire list level through the different itemize/enumerate bullets (‘`-`’, ‘`+`’, ‘`*`’, ‘`1.`’, ‘`1)`’) or a subset of them, depending on `org-plain-list-ordered-item-terminator`, the type of list, and its indentation. With a numeric prefix argument N, select the Nth bullet from this list. If there is an active region when calling this, all lines are converted to list items. With a prefix argument, the selected text is changed into a single item. If the first line already was a list item, any item marker is removed from the list. Finally, even without an active region, a normal line is converted into a list item.

`C-c *` ¶
Turn a plain list item into a headline—so that it becomes a subheading at its location. See Structure Editing, for a detailed explanation.

`C-c C-*` ¶
Turn the whole plain list into a subtree of the current heading. Checkboxes (see Checkboxes) become ‘`TODO`’, respectively ‘`DONE`’, keywords when unchecked, respectively checked.

`S-LEFT` ¶
`S-RIGHT`
This command also cycles bullet styles when point is in on the bullet or anywhere in an item line, details depending on `org-support-shift-select`.

`C-c ^` ¶
Sort the plain list. Prompt for the sorting method: numerically, alphabetically, by time, or by custom function.

2.7 Drawers

Sometimes you want to keep information associated with an entry, but you normally do not want to see it. For this, Org mode has *drawers*. They can contain anything but a headline and another drawer. Drawers look like this:

```
** This is a headline
Still outside the drawer
:DRAWERNAME:
This is inside the drawer.
:END:
After the drawer.
```

You can interactively insert a drawer at point by calling `org-insert-drawer`, which is bound to `C-c C-x d`. With an active region, this command puts the region inside the drawer. With a prefix argument, this command calls non-interactive function `org-insert-property-drawer`, which creates a ‘`PROPERTIES`’ drawer right below the current headline. Org mode uses this special drawer for storing properties (see Properties and Columns). You cannot use it for anything else.

Completion over drawer keywords is also possible using `M-TAB`[16].

Visibility cycling (see Visibility Cycling) on the headline hides and shows the entry, but keep the drawer collapsed to a single line. In order to look inside the drawer, you need to move point to the drawer line and press TAB there.

You can also arrange for state change notes (see Tracking TODO state changes) and clock times (see Clocking Work Time) to be stored in a 'LOGBOOK' drawer. If you want to store a quick note there, in a similar way to state changes, use

C-c C-z ¶
Add a time-stamped note to the 'LOGBOOK' drawer.

2.8 Blocks

Org mode uses '#+BEGIN' … '#+END' blocks for various purposes from including source code examples (see Literal Examples) to capturing time logging information (see Clocking Work Time). These blocks can be folded and unfolded by pressing TAB in the '#+BEGIN' line. You can also get all blocks folded at startup by configuring the variable `org-hide-block-startup` or on a per-file basis by using

```
#+STARTUP: hideblocks
#+STARTUP: nohideblocks
```

3 Tables

Org comes with a fast and intuitive table editor. Spreadsheet-like calculations are supported using the Emacs Calc package (see (calc)GNU Emacs Calculator Manual).

Next: Column Width and Alignment, Up: Tables [Contents][Index]

3.1 Built-in Table Editor

Org makes it easy to format tables in plain ASCII. Any line with '|' as the first non-whitespace character is considered part of a table. '|' is also the column separator[17]. Moreover, a line starting with '|-' is a horizontal rule. It separates rows explicitly. Rows before the first horizontal rule are header lines. A table might look like this:

```
| Name  | Phone | Age |
|-------+-------+-----|
| Peter |  1234 |  17 |
| Anna  |  4321 |  25 |
```

A table is re-aligned automatically each time you press TAB, RET or C-c C-c inside the table. TAB also moves to the next field—RET to the next row—and creates new table rows at the end of the table or before horizontal lines. The indentation of the table is set by the first line. Horizontal rules are automatically expanded on every re-align to span the whole table width. So, to create the above table, you would only type

```
|Name|Phone|Age|
|-
```

and then press TAB to align the table and start filling in fields. Even faster would be to type '|Name|Phone|Age' followed by C-c RET.

When typing text into a field, Org treats `DEL`, `Backspace`, and all character keys in a special way, so that inserting and deleting avoids shifting other fields. Also, when typing *immediately* after point was moved into a new field with `TAB`, `S-TAB` or `RET`, the field is automatically made blank. If this behavior is too unpredictable for you, configure the option `org-table-auto-blank-field`.

Creation and conversion

`C-c |` (**`org-table-create-or-convert-from-region`**) ¶
Convert the active region to table. If every line contains at least one `TAB` character, the function assumes that the material is tab separated. If every line contains a comma, comma-separated values (CSV) are assumed. If not, lines are split at whitespace into fields. You can use a prefix argument to force a specific separator: `C-u` forces CSV, `C-u C-u` forces `TAB`, `C-u C-u C-u` prompts for a regular expression to match the separator, and a numeric argument N indicates that at least N consecutive spaces, or alternatively a `TAB` will be the separator.

If there is no active region, this command creates an empty Org table. But it is easier just to start typing, like `| N a m e | P h o n e | A g e RET | - TAB`.

Re-aligning and field motion

`C-c C-c` (**`org-table-align`**) ¶
Re-align the table without moving point.

`TAB` (**`org-table-next-field`**) ¶
Re-align the table, move to the next field. Creates a new row if necessary.

`M-x org-table-blank-field` ¶
Blank the current table field or active region.

`S-TAB` (**`org-table-previous-field`**) ¶
Re-align, move to previous field.

`RET` (**`org-table-next-row`**) ¶
Re-align the table and move down to next row. Creates a new row if necessary. At the beginning or end of a line, `RET` still inserts a new line, so it can be used to split a table.

`M-a` (**`org-table-beginning-of-field`**) ¶
Move to beginning of the current table field, or on to the previous field.

`M-e` (**`org-table-end-of-field`**) ¶
Move to end of the current table field, or on to the next field.

Column and row editing

`M-LEFT` (**`org-table-move-column-left`**) ¶
Move the current column left.

`M-RIGHT` (**`org-table-move-column-right`**) ¶
Move the current column right.

`M-S-LEFT` (`org-table-delete-column`) ¶
Kill the current column.

`M-S-RIGHT` (`org-table-insert-column`) ¶
Insert a new column at point position. Move the recent column and all cells to the right of this column to the right.

`M-UP` (`org-table-move-row-up`) ¶
Move the current row up.

`M-DOWN` (`org-table-move-row-down`) ¶
Move the current row down.

`M-S-UP` (`org-table-kill-row`) ¶
Kill the current row or horizontal line.

`S-UP` (`org-table-move-cell-up`) ¶
Move cell up by swapping with adjacent cell.

`S-DOWN` (`org-table-move-cell-down`) ¶
Move cell down by swapping with adjacent cell.

`S-LEFT` (`org-table-move-cell-left`) ¶
Move cell left by swapping with adjacent cell.

`S-RIGHT` (`org-table-move-cell-right`) ¶
Move cell right by swapping with adjacent cell.

`M-S-DOWN` (`org-table-insert-row`) ¶
Insert a new row above the current row. With a prefix argument, the line is created below the current one.

`C-c -` (`org-table-insert-hline`) ¶
Insert a horizontal line below current row. With a prefix argument, the line is created above the current line.

`C-c RET` (`org-table-hline-and-move`) ¶
Insert a horizontal line below current row, and move point into the row below that line.

`C-c ^` (`org-table-sort-lines`) ¶
Sort the table lines in the region. The position of point indicates the column to be used for sorting, and the range of lines is the range between the nearest horizontal separator lines, or the entire table. If point is before the first column, you are prompted for the sorting column. If there is an active region, the mark specifies the first line and the sorting column, while point should be in the last line to be included into the sorting. The command prompts for the sorting type, alphabetically, numerically, or by time. You can sort in normal or reverse order. You can also supply your own key extraction and comparison functions. When called with a prefix argument, alphabetic sorting is case-sensitive.

Regions

`C-c C-x M-w` (`org-table-copy-region`) ¶

Copy a rectangular region from a table to a special clipboard. Point and mark determine edge fields of the rectangle. If there is no active region, copy just the current field. The process ignores horizontal separator lines.

`C-c C-x C-w` (`org-table-cut-region`) ¶
Copy a rectangular region from a table to a special clipboard, and blank all fields in the rectangle. So this is the “cut” operation.

`C-c C-x C-y` (`org-table-paste-rectangle`) ¶
Paste a rectangular region into a table. The upper left corner ends up in the current field. All involved fields are overwritten. If the rectangle does not fit into the present table, the table is enlarged as needed. The process ignores horizontal separator lines.

`M-RET` (`org-table-wrap-region`) ¶
Split the current field at point position and move the rest to the line below. If there is an active region, and both point and mark are in the same column, the text in the column is wrapped to minimum width for the given number of lines. A numeric prefix argument may be used to change the number of desired lines. If there is no region, but you specify a prefix argument, the current field is made blank, and the content is appended to the field above.

Calculations

`C-c +` (`org-table-sum`) ¶
Sum the numbers in the current column, or in the rectangle defined by the active region. The result is shown in the echo area and can be inserted with `C-y`.

`S-RET` (`org-table-copy-down`) ¶
When current field is empty, copy from first non-empty field above. When not empty, copy current field down to next row and move point along with it.

Depending on the variable `org-table-copy-increment`, integer and time stamp field values, and fields prefixed or suffixed with a whole number, can be incremented during copy. Also, a `0` prefix argument temporarily disables the increment.

This key is also used by shift-selection and related modes (see Packages that conflict with Org mode).

Miscellaneous

`C-c ` ` (`org-table-edit-field`) ¶
Edit the current field in a separate window. This is useful for fields that are not fully visible (see Column Width and Alignment). When called with a `C-u` prefix, just make the full field visible, so that it can be edited in place. When called with two `C-u` prefixes, make the editor window follow point through the table and always show the current field. The follow mode exits automatically when point leaves the table, or when you repeat this command with `C-u C-u C-c ` `.

`M-x org-table-import` ¶
Import a file as a table. The table should be TAB or whitespace separated. Use, for example, to import a spreadsheet table or data from a database, because these programs generally can write TAB-separated text files. This command works by inserting the file into the buffer and then

converting the region to a table. Any prefix argument is passed on to the converter, which uses it to determine the separator.

`C-c |` (`org-table-create-or-convert-from-region`) ¶
Tables can also be imported by pasting tabular text into the Org buffer, selecting the pasted text with `C-x C-x` and then using the `C-c |` command (see Creation and conversion).

`M-x org-table-export` ¶
Export the table, by default as a TAB-separated file. Use for data exchange with, for example, spreadsheet or database programs. The format used to export the file can be configured in the variable `org-table-export-default-format`. You may also use properties 'TABLE_EXPORT_FILE' and 'TABLE_EXPORT_FORMAT' to specify the file name and the format for table export in a subtree. Org supports quite general formats for exported tables. The exporter format is the same as the format used by Orgtbl radio tables, see Translator functions, for a detailed description.

Turn on the display of the first data row of the table at point in the window header line when this first row is not visible anymore in the buffer. You can activate this minor mode by default by setting the option `org-table-header-line-p` to `t`.

`M-x org-table-transpose-table-at-point` ¶
Transpose the table at point and eliminate hlines.

3.2 Column Width and Alignment

The width of columns is automatically determined by the table editor. The alignment of a column is determined automatically from the fraction of number-like versus non-number fields in the column.

Editing a field may modify alignment of the table. Moving a contiguous row or column—i.e., using `TAB` or `RET`—automatically re-aligns it. If you want to disable this behavior, set `org-table-automatic-realign` to `nil`. In any case, you can always align manually a table:

`C-c C-c` (`org-table-align`) ¶
Align the current table.

Setting the option `org-startup-align-all-tables` re-aligns all tables in a file upon visiting it. You can also set this option on a per-file basis with:

```
#+STARTUP: align
#+STARTUP: noalign
```

Sometimes a single field or a few fields need to carry more text, leading to inconveniently wide columns. Maybe you want to hide away several columns or display them with a fixed width, regardless of content, as shown in the following example.

```
|---+---------------------+--------|           |---+-------…+…|
|   | <6>                 |        |           |   | <6>   …|…|
| 1 | one                 | some   |   ----\   | 1 | one   …|…|
| 2 | two                 | boring |   ----/   | 2 | two   …|…|
| 3 | This is a long text | column |           | 3 | This i…|…|
|---+---------------------+--------|           |---+-------…+…|
```

To set the width of a column, one field anywhere in the column may contain just the string '<N>' where *N* specifies the width as a number of characters. You control displayed width of columns with the following tools:

C-c TAB (**org-table-toggle-column-width**) ¶
Shrink or expand current column.

If a width cookie specifies a width W for the column, shrinking it displays the first W visible characters only. Otherwise, the column is shrunk to a single character.

When called before the first column or after the last one, ask for a list of column ranges to operate on.

C-u C-c TAB (**org-table-shrink**) ¶
Shrink all columns with a column width. Expand the others.

C-u C-u C-c TAB (**org-table-expand**) ¶
Expand all columns.

To see the full text of a shrunk field, hold the mouse over it: a tool-tip window then shows the full contents of the field. Alternatively, `C-h .` (`display-local-help`) reveals them, too. For convenience, any change near the shrunk part of a column expands it.

Setting the option `org-startup-shrink-all-tables` shrinks all columns containing a width cookie in a file the moment it is visited. You can also set this option on a per-file basis with:

```
#+STARTUP: shrink
```

If you would like to overrule the automatic alignment of number-rich columns to the right and of string-rich columns to the left, you can use '<r>', '<c>' or '<l>' in a similar fashion. You may also combine alignment and field width like this: '<r10>'.

Lines which only contain these formatting cookies are removed automatically upon exporting the document.

3.3 Column Groups

When Org exports tables, it does so by default without vertical lines because that is visually more satisfying in general. Occasionally however, vertical lines can be useful to structure a table into groups of columns, much like horizontal lines can do for groups of rows. In order to specify column groups, you can use a special row where the first field contains only '/'. The further fields can either contain '<' to indicate that this column should start a group, '>' to indicate the end of a column, or '<>' (no space between '<' and '>') to make a column a group of its own. Upon export, boundaries between column groups are marked with vertical lines. Here is an example:

```
| N | N^2 | N^3 | N^4 | sqrt(n) | sqrt[4](N) |
|---+-----+-----+-----+---------+------------|
| / |  <  |     |  >  |       < |          > |
| 1 |  1  |  1  |  1  |       1 |          1 |
| 2 |  4  |  8  | 16  |  1.4142 |     1.1892 |
| 3 |  9  | 27  | 81  |  1.7321 |     1.3161 |
|---+-----+-----+-----+---------+------------|
#+TBLFM: $2=$1^2::$3=$1^3::$4=$1^4::$5=sqrt($1)::$6=sqrt(sqrt(($1)))
```

It is also sufficient to just insert the column group starters after every vertical line you would like to have:

```
| N | N^2 | N^3 | N^4 | sqrt(n) | sqrt[4](N) |
|---+-----+-----+-----+---------+------------|
| / | <   |     |     | <       |            |
```

3.4 The Orgtbl Minor Mode

If you like the intuitive way the Org table editor works, you might also want to use it in other modes like Text mode or Mail mode. The minor mode Orgtbl mode makes this possible. You can always toggle the mode with `M-x orgtbl-mode`. To turn it on by default, for example in Message mode, use

```
(add-hook 'message-mode-hook #'turn-on-orgtbl)
```

Furthermore, with some special setup, it is possible to maintain tables in arbitrary syntax with Orgtbl mode. For example, it is possible to construct LaTeX tables with the underlying ease and power of Orgtbl mode, including spreadsheet capabilities. For details, see Tables in Arbitrary Syntax.

3.5 The Spreadsheet

The table editor makes use of the Emacs Calc package to implement spreadsheet-like capabilities. It can also evaluate Emacs Lisp forms to derive fields from other fields. While fully featured, Org's implementation is not identical to other spreadsheets. For example, Org knows the concept of a *column formula* that will be applied to all non-header fields in a column without having to copy the formula to each relevant field. There is also a formula debugger, and a formula editor with features for highlighting fields in the table corresponding to the references at point in the formula, moving these references by arrow keys.

Next: Formula syntax for Calc, Up: The Spreadsheet [Contents][Index]

3.5.1 References

To compute fields in the table from other fields, formulas must reference other fields or ranges. In Org, fields can be referenced by name, by absolute coordinates, and by relative coordinates. To find out what the coordinates of a field are, press `C-c ?` in that field, or press `C-c }` to toggle the display of a grid.

Field references

Formulas can reference the value of another field in two ways. Like in any other spreadsheet, you may reference fields with a letter/number combination like 'B3', meaning the second field in the third row. However, Org prefers to use another, more general representation that looks like

this:[18]

```
@ROW$COLUMN
```

Column specifications can be absolute like '$1', '$2', …, '$N', or relative to the current column, i.e., the column of the field which is being computed, like '$+1' or '$-2'. '$<' and '$>' are immutable references to the first and last column, respectively, and you can use '$>>>' to indicate the third column from the right.

The row specification only counts data lines and ignores horizontal separator lines, or "hlines". Like with columns, you can use absolute row numbers '@1', '@2', …, '@N', and row numbers relative to the current row like '@+3' or '@-1'. '@<' and '@>' are immutable references the first and last row in the table, respectively. You may also specify the row relative to one of the hlines: '@I' refers to the first hline, '@II' to the second, etc. '@-I' refers to the first such line above the current line, '@+I' to the first such line below the current line. You can also write '@III+2' which is the second data line after the third hline in the table.

'@0' and '$0' refer to the current row and column, respectively, i.e., to the row/column for the field being computed. Also, if you omit either the column or the row part of the reference, the current row/column is implied.

Org's references with *unsigned* numbers are fixed references in the sense that if you use the same reference in the formula for two different fields, the same field is referenced each time. Org's references with *signed* numbers are floating references because the same reference operator can reference different fields depending on the field being calculated by the formula.

Here are a few examples:

'@2$3'	2nd row, 3rd column (same as 'C2')
'$5'	column 5 in the current row (same as 'E&')
'@2'	current column, row 2
'@-1$-3'	field one row up, three columns to the left
'@-I$2'	field just under hline above current row, column 2
'@>$5'	field in the last row, in column 5

Range references

You may reference a rectangular range of fields by specifying two field references connected by two dots '..'. The ends are included in the range. If both fields are in the current row, you may simply use '$2..$7', but if at least one field is in a different row, you need to use the general '@ROW$COLUMN' format at least for the first field, i.e., the reference must start with '@' in order to be interpreted correctly. Examples:

'$1..$3'	first three fields in the current row
'$P..$Q'	range, using column names (see Advanced features)

'$<<<..$>>'	start in third column, continue to the last but one
'@2$1..@4$3'	nine fields between these two fields (same as 'A2..C4')
'@-1$-2..@-1'	3 fields in the row above, starting from 2 columns on the left
'@I..II'	between first and second hline, short for '@I..@II'

Range references return a vector of values that can be fed into Calc vector functions. Empty fields in ranges are normally suppressed, so that the vector contains only the non-empty fields. For other options with the mode switches 'E', 'N' and examples, see Formula syntax for Calc.

Field coordinates in formulas

One of the very first actions during evaluation of Calc formulas and Lisp formulas is to substitute '@#' and '$#' in the formula with the row or column number of the field where the current result will go to. The traditional Lisp formula equivalents are `org-table-current-dline` and `org-table-current-column`. Examples:

'if(@# % 2, $#, string(""))'
Insert column number on odd rows, set field to empty on even rows.

'$2 = '(identity remote(FOO, @@#$1))'
Copy text or values of each row of column 1 of the table named *FOO* into column 2 of the current table.

'@3 = 2 * remote(FOO, @1$$#)'
Insert the doubled value of each column of row 1 of the table named *FOO* into row 3 of the current table.

For the second and third examples, table *FOO* must have at least as many rows or columns as the current table. Note that this is inefficient[19] for large number of rows.

Named references

'$name' is interpreted as the name of a column, parameter or constant. Constants are defined globally through the variable `org-table-formula-constants`, and locally—for the file—through a line like this example:

```
#+CONSTANTS: c=299792458. pi=3.14 eps=2.4e-6
```

Also, properties (see Properties and Columns) can be used as constants in table formulas: for a property 'Xyz' use the name '$PROP_Xyz', and the property will be searched in the current outline entry and in the hierarchy above it. If you have the '`constants.el`' package, it will also be used to resolve constants, including natural constants like '$h' for Planck's constant, and units like '$km' for kilometers[20]. Column names and parameters can be specified in special table lines. These are described below, see Advanced features. All names must start with a letter, and further consist of letters and numbers.

Remote references

You may also reference constants, fields and ranges from a different table, either in the current file or even in a different file. The syntax is

```
remote(NAME,REF)
```

where *NAME* can be the name of a table in the current file as set by a '`#+NAME:`' line before the table. It can also be the ID of an entry, even in a different file, and the reference then refers to the first table in that entry. *REF* is an absolute field or range reference as described above for example '`@3$3`' or '`$somename`', valid in the referenced table.

When *NAME* has the format '`@ROW$COLUMN`', it is substituted with the name or ID found in this field of the current table. For example '`remote($1, @@>$2)`' ⇒ '`remote(year_2013, @@>$1)`'. The format '`B3`' is not supported because it can not be distinguished from a plain table name or ID.

3.5.2 Formula syntax for Calc

A formula can be any algebraic expression understood by the Emacs Calc package. Note that Calc has the non-standard convention that '`/`' has lower precedence than '`*`', so that '`a/b*c`' is interpreted as '`(a/(b*c))`'. Before evaluation by `calc-eval` (see (calc)Calling Calc from Your Lisp Programs), variable substitution takes place according to the rules described above.

The range vectors can be directly fed into the Calc vector functions like `vmean` and `vsum`.

A formula can contain an optional mode string after a semicolon. This string consists of flags to influence Calc and other modes during execution. By default, Org uses the standard Calc modes (precision 12, angular units degrees, fraction and symbolic modes off). The display format, however, has been changed to '`(float 8)`' to keep tables compact. The default settings can be configured using the variable `org-calc-default-modes`.

'`p20`'
Set the internal Calc calculation precision to 20 digits.

'`n3`', '`s3`', '`e2`', '`f4`'
Normal, scientific, engineering or fixed format of the result of Calc passed back to Org. Calc formatting is unlimited in precision as long as the Calc calculation precision is greater.

'`D`', '`R`'
Degree and radian angle modes of Calc.

'`F`', '`S`'
Fraction and symbolic modes of Calc.

'`u`'
Units simplification mode of Calc. Calc is also a symbolic calculator and is capable of working with values having a unit, represented with numerals followed by a unit string in Org table cells. This mode instructs Calc to simplify the units in the computed expression before returning the result.

'`T`', '`t`', '`U`'
Duration computations in Calc or Lisp, Durations and time values.

‘E’
If and how to consider empty fields. Without ‘E’ empty fields in range references are suppressed so that the Calc vector or Lisp list contains only the non-empty fields. With ‘E’ the empty fields are kept. For empty fields in ranges or empty field references the value ‘nan’ (not a number) is used in Calc formulas and the empty string is used for Lisp formulas. Add ‘N’ to use 0 instead for both formula types. For the value of a field the mode ‘N’ has higher precedence than ‘E’.

‘N’
Interpret all fields as numbers, use 0 for non-numbers. See the next section to see how this is essential for computations with Lisp formulas. In Calc formulas it is used only occasionally because there number strings are already interpreted as numbers without ‘N’.

‘L’
Literal, for Lisp formulas only. See the next section.

Unless you use large integer numbers or high-precision calculation and display for floating point numbers you may alternatively provide a `printf` format specifier to reformat the Calc result after it has been passed back to Org instead of letting Calc already do the formatting[21]. A few examples:

‘$1+$2’	Sum of first and second field
‘$1+$2;%.2f’	Same, format result to two decimals
‘exp($2)+exp($1)’	Math functions can be used
‘$0;%.1f’	Reformat current cell to 1 decimal
‘($3-32)*5/9’	Degrees F → C conversion
‘$c/$1/$cm’	Hz → cm conversion, using ‘constants.el’
‘tan($1);Dp3s1’	Compute in degrees, precision 3, display SCI 1
‘sin($1);Dp3%.1e’	Same, but use `printf` specifier for display
‘vmean($2..$7)’	Compute column range mean, using vector function
‘vmean($2..$7);EN’	Same, but treat empty fields as 0
‘taylor($3,x=7,2)’	Taylor series of $3, at x=7, second degree

Calc also contains a complete set of logical operations (see (calc)Logical Operations). For example

‘if($1 < 20, teen, string(""))’
‘"teen"’ if age ‘$1’ is less than 20, else the Org table result field is set to empty with the empty string.

‘if("$1" == "nan" || "$2" == "nan", string(""), $1 + $2); E f-1’

Sum of the first two columns. When at least one of the input fields is empty the Org table result field is set to empty. ‘E’ is required to not convert empty fields to 0. ‘`f-1`’ is an optional Calc format string similar to ‘`%.1f`’ but leaves empty results empty.

‘`if(typeof(vmean($1..$7)) == 12, string(""), vmean($1..$7)); E`’
Mean value of a range unless there is any empty field. Every field in the range that is empty is replaced by ‘`nan`’ which lets ‘`vmean`’ result in ‘`nan`’. Then ‘`typeof ==`’ 12= detects the ‘`nan`’ from `vmean` and the Org table result field is set to empty. Use this when the sample set is expected to never have missing values.

‘`if("$1..$7" == "[]", string(""), vmean($1..$7))`’
Mean value of a range with empty fields skipped. Every field in the range that is empty is skipped. When all fields in the range are empty the mean value is not defined and the Org table result field is set to empty. Use this when the sample set can have a variable size.

‘`vmean($1..$7); EN`’
To complete the example before: Mean value of a range with empty fields counting as samples with value 0. Use this only when incomplete sample sets should be padded with 0 to the full size.

You can add your own Calc functions defined in Emacs Lisp with `defmath` and use them in formula syntax for Calc.

3.5.3 Emacs Lisp forms as formulas

It is also possible to write a formula in Emacs Lisp. This can be useful for string manipulation and control structures, if Calc’s functionality is not enough.

A formula is evaluated as a Lisp form when it starts with a single-quote followed by an opening parenthesis. Cell table references are interpolated into the Lisp form before execution. The evaluation should return either a string or a number. Evaluation modes and a `printf` format used to render the returned values can be specified after a semicolon.

By default, references are interpolated as literal Lisp strings: the field content is replaced in the Lisp form stripped of leading and trailing white space and surrounded in double-quotes. For example:

```
'(concat $1 $2)
```

concatenates the content of columns 1 and column 2.

When the ‘`N`’ flag is used, all referenced elements are parsed as numbers and interpolated as Lisp numbers, without quotes. Fields that cannot be parsed as numbers are interpolated as zeros. For example:

```
'(+ $1 $2);N
```

adds columns 1 and 2, equivalent to Calc’s ‘$1+$2’. Ranges are inserted as space-separated fields, so they can be embedded in list or vector syntax. For example:

```
'(apply '+ '($1..$4));N
```

computes the sum of columns 1 to 4, like Calc's '`vsum($1..$4)`'.

When the '`L`' flag is used, all fields are interpolated literally: the cell content is replaced in the Lisp form stripped of leading and trailing white space and without quotes. If a reference is intended to be interpreted as a string by the Lisp form, the reference operator itself should be enclosed in double-quotes, like '`"$3"`'. The '`L`' flag is useful when strings and numbers are used in the same Lisp form. For example:

```
'(substring "$1" $2 $3);L
```

extracts the part of the string in column 1 between the character positions specified in the integers in column 2 and 3 and it is easier to read than the equivalent:

```
'(substring $1 (string-to-number $2) (string-to-number $3))
```

When the formula itself contains '`;`' symbol, Org mode may incorrectly interpret everything past '`;`' as format specifier:

```
'(concat $1 ";")
```

You can put an extra tailing '`;`' to indicate that all the earlier instances of '`;`' belong to the formula itself:

```
'(concat $1 ";");
```

3.5.4 Durations and time values

If you want to compute time values use the '`T`', '`t`', or '`U`' flag, either in Calc formulas or Elisp formulas:

```
|  Task 1 |   Task 2 |    Total |
|---------+----------+----------|
|    2:12 |     1:47 | 03:59:00 |
|    2:12 |     1:47 |    03:59 |
| 3:02:20 | -2:07:00 |     0.92 |
#+TBLFM: @2$3=$1+$2;T::@3$3=$1+$2;U::@4$3=$1+$2;t
```

Input duration values must be of the form '`HH:MM[:SS]`', where seconds are optional. With the '`T`' flag, computed durations are displayed as '`HH:MM:SS`' (see the first formula above). With the '`U`' flag, seconds are omitted so that the result is only '`HH:MM`' (see second formula above). Zero-padding of the hours field depends upon the value of the variable `org-table-duration-hour-zero-padding`.

With the '`t`' flag, computed durations are displayed according to the value of the option `org-table-duration-custom-format`, which defaults to `hours` and displays the result as a fraction of hours (see the third formula in the example above).

Negative duration values can be manipulated as well, and integers are considered as seconds in addition and subtraction.

3.5.5 Field and range formulas

To assign a formula to a particular field, type it directly into the field, preceded by ‘:=’, for example ‘vsum(@II..III)’. When you press TAB or RET or C-c C-c with point still in the field, the formula is stored as the formula for this field, evaluated, and the current field is replaced with the result.

Formulas are stored in a special ‘TBLFM’ keyword located directly below the table. If you type the equation in the fourth field of the third data line in the table, the formula looks like ‘@3$4=$1+$2’. When inserting/deleting/swapping column and rows with the appropriate commands, *absolute references* (but not relative ones) in stored formulas are modified in order to still reference the same field. To avoid this from happening, in particular in range references, anchor ranges at the table borders (using ‘@<’, ‘@>’, ‘$<’, ‘$>’), or at hlines using the ‘@I’ notation. Automatic adaptation of field references does not happen if you edit the table structure with normal editing commands—you must fix the formulas yourself.

Instead of typing an equation into the field, you may also use the following command

C-u C-c = (org-table-eval-formula) ¶
Install a new formula for the current field. The command prompts for a formula with default taken from the ‘TBLFM’ keyword, applies it to the current field, and stores it.

The left-hand side of a formula can also be a special expression in order to assign the formula to a number of different fields. There is no keyboard shortcut to enter such range formulas. To add them, use the formula editor (see Editing and debugging formulas) or edit the ‘TBLFM’ keyword directly.

‘$2=’
Column formula, valid for the entire column. This is so common that Org treats these formulas in a special way, see Column formulas.

‘@3=’
Row formula, applies to all fields in the specified row. ‘@>=’ means the last row.

‘@1$2..@4$3=’
Range formula, applies to all fields in the given rectangular range. This can also be used to assign a formula to some but not all fields in a row.

‘$NAME=’
Named field, see Advanced features.

3.5.6 Column formulas

When you assign a formula to a simple column reference like ‘$3=’, the same formula is used in all fields of that column, with the following very convenient exceptions: (i) If the table contains horizontal separator hlines with rows above and below, everything before the first such hline is considered part of the table *header* and is not modified by column formulas. Therefore a header is mandatory when you use column formulas and want to add hlines to group rows, like for example to separate a total row at the bottom from the summand rows above. (ii) Fields that already get a value from a field/range formula are left alone by column formulas. These conditions make column formulas very easy to use.

To assign a formula to a column, type it directly into any field in the column, preceded by an equal sign, like '=$1+$2'. When you press `TAB` or `RET` or `C-c C-c` with point still in the field, the formula is stored as the formula for the current column, evaluated and the current field replaced with the result. If the field contains only '=', the previously stored formula for this column is used. For each column, Org only remembers the most recently used formula. In the 'TBLFM' keyword, column formulas look like '$4=$1+$2'. The left-hand side of a column formula can not be the name of column, it must be the numeric column reference or '$>'.

Instead of typing an equation into the field, you may also use the following command:

`C-c =` (`org-table-eval-formula`) ¶
Install a new formula for the current column and replace current field with the result of the formula. The command prompts for a formula, with default taken from the 'TBLFM' keyword, applies it to the current field and stores it. With a numeric prefix argument, e.g., `C-5 C-c =`, the command applies it to that many consecutive fields in the current column.

3.5.7 Lookup functions

Org has three predefined Emacs Lisp functions for lookups in tables.

'(org-lookup-first VAL S-LIST R-LIST &optional PREDICATE)' ¶
Searches for the first element *S* in list *S-LIST* for which

```
(PREDICATE VAL S)
```

is non-`nil`; returns the value from the corresponding position in list *R-LIST*. The default *PREDICATE* is `equal`. Note that the parameters *VAL* and *S* are passed to *PREDICATE* in the same order as the corresponding parameters are in the call to `org-lookup-first`, where *VAL* precedes *S-LIST*. If *R-LIST* is `nil`, the matching element *S* of *S-LIST* is returned.

'(org-lookup-last VAL S-LIST R-LIST &optional PREDICATE)' ¶
Similar to `org-lookup-first` above, but searches for the *last* element for which *PREDICATE* is non-`nil`.

'(org-lookup-all VAL S-LIST R-LIST &optional PREDICATE)' ¶
Similar to `org-lookup-first`, but searches for *all* elements for which *PREDICATE* is non-`nil`, and returns *all* corresponding values. This function can not be used by itself in a formula, because it returns a list of values. However, powerful lookups can be built when this function is combined with other Emacs Lisp functions.

If the ranges used in these functions contain empty fields, the 'E' mode for the formula should usually be specified: otherwise empty fields are not included in *S-LIST* and/or *R-LIST* which can, for example, result in an incorrect mapping from an element of *S-LIST* to the corresponding element of *R-LIST*.

These three functions can be used to implement associative arrays, count matching cells, rank results, group data, etc. For practical examples see this tutorial on Worg.

Next: Updating the table, Previous: Lookup functions, Up: The Spreadsheet [Contents][Index]

3.5.8 Editing and debugging formulas

You can edit individual formulas in the minibuffer or directly in the field. Org can also prepare a special buffer with all active formulas of a table. When offering a formula for editing, Org converts references to the standard format (like ‘B3’ or ‘D&’) if possible. If you prefer to only work with the internal format (like ‘@3$2’ or ‘$4’), configure the variable `org-table-use-standard-references`.

`C-c =` or **`C-u C-c =`** (**`org-table-eval-formula`**) ¶
Edit the formula associated with the current column/field in the minibuffer. See Column formulas, and Field and range formulas.

`C-u C-u C-c =` (**`org-table-eval-formula`**) ¶
Re-insert the active formula (either a field formula, or a column formula) into the current field, so that you can edit it directly in the field. The advantage over editing in the minibuffer is that you can use the command `C-c ?`.

`C-c ?` (**`org-table-field-info`**) ¶
While editing a formula in a table field, highlight the field(s) referenced by the reference at point position in the formula.

`C-c }` (**`org-table-toggle-coordinate-overlays`**) ¶
Toggle the display of row and column numbers for a table, using overlays. These are updated each time the table is aligned; you can force it with `C-c C-c`.

`C-c {` (**`org-table-toggle-formula-debugger`**) ¶
Toggle the formula debugger on and off. See below.

`C-c '` (**`org-table-edit-formulas`**) ¶
Edit all formulas for the current table in a special buffer, where the formulas are displayed one per line. If the current field has an active formula, point in the formula editor marks it. While inside the special buffer, Org automatically highlights any field or range reference at point position. You may edit, remove and add formulas, and use the following commands:

`C-c C-c` or **`C-x C-s`** (**`org-table-fedit-finish`**) ¶
Exit the formula editor and store the modified formulas. With `C-u` prefix, also apply the new formulas to the entire table.

`C-c C-q` (**`org-table-fedit-abort`**) ¶
Exit the formula editor without installing changes.

`C-c C-r` (**`org-table-fedit-toggle-ref-type`**) ¶
Toggle all references in the formula editor between standard (like ‘B3’) and internal (like ‘@3$2’).

`TAB` (**`org-table-fedit-lisp-indent`**) ¶
Pretty-print or indent Lisp formula at point. When in a line containing a Lisp formula, format the formula according to Emacs Lisp rules. Another `TAB` collapses the formula back again. In the open formula, `TAB` re-indents just like in Emacs Lisp mode.

`M-TAB` (**`lisp-complete-symbol`**) ¶
Complete Lisp symbols, just like in Emacs Lisp mode.

`S-UP`, **`S-DOWN`**, **`S-LEFT`**, **`S-RIGHT`** ¶

Shift the reference at point. For example, if the reference is ‘B3’ and you press `S-RIGHT`, it becomes ‘C3’. This also works for relative references and for hline references.

`M-S-UP` (`org-table-fedit-line-up`) ¶
Move the test line for column formulas up in the Org buffer.

`M-S-DOWN` (`org-table-fedit-line-down`) ¶
Move the test line for column formulas down in the Org buffer.

`M-UP` (`org-table-fedit-scroll-up`) ¶
Scroll up the window displaying the table.

`M-DOWN` (`org-table-fedit-scroll-down`) ¶
Scroll down the window displaying the table.

`C-c }` ¶
Turn the coordinate grid in the table on and off.

Making a table field blank does not remove the formula associated with the field, because that is stored in a different line—the ‘TBLFM’ keyword line. During the next recalculation, the field will be filled again. To remove a formula from a field, you have to give an empty reply when prompted for the formula, or to edit the ‘TBLFM’ keyword.

You may edit the ‘TBLFM’ keyword directly and re-apply the changed equations with `C-c C-c` in that line or with the normal recalculation commands in the table.

Using multiple ‘TBLFM’ lines

You may apply the formula temporarily. This is useful when you want to switch the formula applied to the table. Place multiple ‘TBLFM’ keywords right after the table, and then press `C-c C-c` on the formula to apply. Here is an example:

```
| x | y |
|---+---|
| 1 |   |
| 2 |   |
#+TBLFM: $2=$1*1
#+TBLFM: $2=$1*2
```

Pressing `C-c C-c` in the line of ‘#+TBLFM: $2=$1*2’ yields:

```
| x | y |
|---+---|
| 1 | 2 |
| 2 | 4 |
#+TBLFM: $2=$1*1
#+TBLFM: $2=$1*2
```

If you recalculate this table, with `C-u C-c *`, for example, you get the following result from applying only the first ‘TBLFM’ keyword.

```
| x | y |
|---+---|
| 1 | 1 |
| 2 | 2 |
#+TBLFM: $2=$1*1
#+TBLFM: $2=$1*2
```

Debugging formulas

When the evaluation of a formula leads to an error, the field content becomes the string '#ERR0R'. If you want to see what is going on during variable substitution and calculation in order to find a bug, turn on formula debugging in the Tbl menu and repeat the calculation, for example by pressing `C-u C-u C-c = RET` in a field. Detailed information are displayed.

3.5.9 Updating the table

Recalculation of a table is normally not automatic, but needs to be triggered by a command. To make recalculation at least semi-automatic, see Advanced features.

In order to recalculate a line of a table or the entire table, use the following commands:

`C-c *` (`org-table-recalculate`) ¶
Recalculate the current row by first applying the stored column formulas from left to right, and all field/range formulas in the current row.

`C-u C-c *` or **`C-u C-c C-c`** ¶
Recompute the entire table, line by line. Any lines before the first hline are left alone, assuming that these are part of the table header.

`C-u C-u C-c *` or **`C-u C-u C-c C-c`** (`org-table-iterate`) ¶
Iterate the table by recomputing it until no further changes occur. This may be necessary if some computed fields use the value of other fields that are computed *later* in the calculation sequence.

`M-x org-table-recalculate-buffer-tables` ¶
Recompute all tables in the current buffer.

`M-x org-table-iterate-buffer-tables` ¶
Iterate all tables in the current buffer, in order to converge table-to-table dependencies.

3.5.10 Advanced features

If you want the recalculation of fields to happen automatically, or if you want to be able to assign *names*[22] to fields and columns, you need to reserve the first column of the table for special marking characters.

`C-#` (`org-table-rotate-recalc-marks`) ¶
Rotate the calculation mark in first column through the states '#', '*', '!', '$'. When there is an active region, change all marks in the region.

Here is an example of a table that collects exam results of students and makes use of these features:

```
|---+---------+--------+--------+--------+-------+------|
|   | Student | Prob 1 | Prob 2 | Prob 3 | Total | Note |
|---+---------+--------+--------+--------+-------+------|
| ! |         |     P1 |     P2 |     P3 |   Tot |      |
| # | Maximum |     10 |     15 |     25 |    50 | 10.0 |
| ^ |         |     m1 |     m2 |     m3 |    mt |      |
|---+---------+--------+--------+--------+-------+------|
| # | Peter   |     10 |      8 |     23 |    41 |  8.2 |
| # | Sam     |      2 |      4 |      3 |     9 |  1.8 |
|---+---------+--------+--------+--------+-------+------|
|   | Average |        |        |        |  25.0 |      |
| ^ |         |        |        |        |    at |      |
| $ | max=50  |        |        |        |       |      |
|---+---------+--------+--------+--------+-------+------|
#+TBLFM: $6=vsum($P1..$P3)::$7=10*$Tot/$max;%.1f::$at=vmean(@-II..@-I);%.1f
```

> **Important:** Please note that for these special tables, recalculating the table with `C-u C-c *` only affects rows that are marked '#' or '*', and fields that have a formula assigned to the field itself. The column formulas are not applied in rows with empty first field.

The marking characters have the following meaning:

'!'
The fields in this line define names for the columns, so that you may refer to a column as '`$Tot`' instead of '`$6`'.

'^'
This row defines names for the fields *above* the row. With such a definition, any formula in the table may use '`$m1`' to refer to the value '`10`'. Also, if you assign a formula to a names field, it is stored as '`$name = ...`'.

'_'
Similar to '^', but defines names for the fields in the row *below*.

'$'
Fields in this row can define *parameters* for formulas. For example, if a field in a '$' row contains '`max=50`', then formulas in this table can refer to the value 50 using '`$max`'. Parameters work exactly like constants, only that they can be defined on a per-table basis.

'#'
Fields in this row are automatically recalculated when pressing `TAB` or `RET` or `S-TAB` in this row. Also, this row is selected for a global recalculation with `C-u C-c *`. Unmarked lines are left alone by this command.

'*'
Selects this line for global recalculation with `C-u C-c *`, but not for automatic recalculation. Use this when automatic recalculation slows down editing too much.

'/'
Do not export this line. Useful for lines that contain the narrowing '<N>' markers or column group markers.

Finally, just to whet your appetite for what can be done with the fantastic Calc package, here is a table that computes the Taylor series of degree n at location x for a couple of functions.

```
|---+-------------+---+-----+-------------------------------------|
|   | Func        | n | x   | Result                              |
|---+-------------+---+-----+-------------------------------------|
| # | exp(x)      | 1 | x   | 1 + x                               |
| # | exp(x)      | 2 | x   | 1 + x + x^2 / 2                     |
| # | exp(x)      | 3 | x   | 1 + x + x^2 / 2 + x^3 / 6           |
| # | x^2+sqrt(x) | 2 | x=0 | x*(0.5 / 0) + x^2 (2 - 0.25 / 0) / 2 |
| # | x^2+sqrt(x) | 2 | x=1 | 2 + 2.5 x - 2.5 + 0.875 (x - 1)^2   |
| * | tan(x)      | 3 | x   | 0.0175 x + 1.77e-6 x^3              |
|---+-------------+---+-----+-------------------------------------|
#+TBLFM: $5=taylor($2,$4,$3);n3
```

Previous: The Spreadsheet, Up: Tables [Contents][Index]

3.6 Org Plot

Org Plot can produce graphs of information stored in Org tables, either graphically or in ASCII art.

Graphical plots using Gnuplot

Org Plot can produce 2D and 3D graphs of information stored in Org tables using Gnuplot and Gnuplot mode. To see this in action, ensure that you have both Gnuplot and Gnuplot mode installed on your system, then call `C-c " g` or `M-x org-plot/gnuplot` on the following table.

```
#+PLOT: title:"Citas" ind:1 deps:(3) type:2d with:histograms set:"yrange [0:]"
| Sede      | Max cites | H-index |
|-----------+-----------+---------|
| Chile     |    257.72 |   21.39 |
| Leeds     |    165.77 |   19.68 |
| Sao Paolo |     71.00 |   11.50 |
| Stockholm |    134.19 |   14.33 |
| Morelia   |    257.56 |   17.67 |
```

Org Plot supports a range of plot types, and provides the ability to add more. For example, a radar plot can be generated like so:

```
#+PLOT: title:"An evaluation of plaintext document formats" transpose:yes type:radar
min:0 max:4
| Format            | Fine-grained-control | Initial Effort | Syntax simplicity | Editor
Support | Integrations | Ease-of-referencing | Versatility |
|-------------------+----------------------+----------------+-------------------+-------
---------+--------------+---------------------+-------------|
| Word              |                    2 |              4 |                 4 |
2 |            3 |                   2 |           2 |
| LaTeX             |                    4 |              1 |                 1 |
3 |            2 |                   4 |           3 |
| Org Mode          |                    4 |              2 |               3.5 |
1 |            4 |                   4 |           4 |
| Markdown          |                    1 |              3 |                 3 |
4 |            3 |                   3 |           1 |
| Markdown + Pandoc |                  2.5 |            2.5 |               2.5 |
3 |            3 |                   3 |           2 |
```

Notice that Org Plot is smart enough to apply the table's headers as labels. Further control over the labels, type, content, and appearance of plots can be exercised through the '`PLOT`' keyword preceding a table. See below for a complete list of Org Plot options. For more information and examples see the Org Plot tutorial.

Plot options

'`set`'
Specify any Gnuplot option to be set when graphing.

'`title`'
Specify the title of the plot.

'`ind`'
Specify which column of the table to use as the '`x`' axis.

'`timeind`'
Specify which column of the table to use as the '`x`' axis as a time value.

'`deps`'
Specify the columns to graph as a Lisp style list, surrounded by parentheses and separated by spaces for example '`dep:(3 4)`' to graph the third and fourth columns. Defaults to graphing all other columns aside from the '`ind`' column.

'`transpose`'
When '`y`', '`yes`', or '`t`' attempt to transpose the table data before plotting. Also recognizes the shorthand option '`trans`'.

'`type`'
Specify the type of the plot, by default one of '`2d`', '`3d`', '`radar`', or '`grid`'. Available types can be customized with `org-plot/preset-plot-types`.

'`with`'
Specify a '`with`' option to be inserted for every column being plotted, e.g., '`lines`', '`points`', '`boxes`', '`impulses`'. Defaults to '`lines`'.

'`file`'
If you want to plot to a file, specify '`"path/to/desired/output-file"`'.

'`labels`'
List of labels to be used for the '`deps`'. Defaults to the column headers if they exist.

'`line`'
Specify an entire line to be inserted in the Gnuplot script.

'`map`'
When plotting '`3d`' or '`grid`' types, set this to '`t`' to graph a flat mapping rather than a '`3d`' slope.

'`min`'

Provides a minimum axis value that may be used by a plot type. Implicitly assumes the ‘y’ axis is being referred to. Can explicitly provide a value for a either the ‘x’ or ‘y’ axis with ‘`xmin`’ and ‘`ymin`’.

‘`max`’
Provides a maximum axis value that may be used by a plot type. Implicitly assumes the ‘y’ axis is being referred to. Can explicitly provide a value for a either the ‘x’ or ‘y’ axis with ‘`xmax`’ and ‘`ymax`’.

‘`ticks`’
Provides a desired number of axis ticks to display, that may be used by a plot type. If none is given a plot type that requires ticks will use `org--plot/sensible-tick-num` to try to determine a good value.

‘`timefmt`’
Specify format of Org mode timestamps as they will be parsed by Gnuplot. Defaults to ‘`%Y-%m-%d-%H:%M:%S`’.

‘`script`’
If you want total control, you can specify a script file—place the file name between double-quotes—which will be used to plot. Before plotting, every instance of ‘`$datafile`’ in the specified script will be replaced with the path to the generated data file. Note: even if you set this option, you may still want to specify the plot type, as that can impact the content of the data file.

ASCII bar plots

While point is on a column, typing `C-c " a` or `M-x orgtbl-ascii-plot` create a new column containing an ASCII-art bars plot. The plot is implemented through a regular column formula. When the source column changes, the bar plot may be updated by refreshing the table, for example typing `C-u C-c *`.

```
| Sede          | Max cites |              |
|---------------+-----------+--------------|
| Chile         |    257.72 | WWWWWWWWWWW  |
| Leeds         |    165.77 | WWWWWWWh     |
| Sao Paolo     |     71.00 | WWW;         |
| Stockholm     |    134.19 | WWWWWW:      |
| Morelia       |    257.56 | WWWWWWWWWWH  |
| Rochefourchat |      0.00 |              |
#+TBLFM: $3='(orgtbl-ascii-draw $2 0.0 257.72 12)
```

The formula is an Elisp call.

Function: orgtbl-ascii-draw *value min max &optional width* ¶
Draw an ASCII bar in a table.

VALUE is the value to plot.

MIN is the value displayed as an empty bar. *MAX* is the value filling all the *WIDTH*. Sources values outside this range are displayed as ‘`too small`’ or ‘`too large`’.

WIDTH is the number of characters of the bar plot. It defaults to ‘12’.

4 Hyperlinks

Like HTML, Org provides support for links inside a file, external links to other files, Usenet articles, emails, and much more.

4.1 Link Format

Org recognizes plain URIs, possibly wrapped within angle brackets[23], and activate them as clickable links.

The general link format, however, looks like this:

```
[[LINK][DESCRIPTION]]
```

or alternatively

```
[[LINK]]
```

Some ‘\’, ‘[’ and ‘]’ characters in the *LINK* part need to be “escaped”, i.e., preceded by another ‘\’ character. More specifically, the following characters, and only them, must be escaped:

1. all ‘[’ and ‘]’ characters,
2. every ‘\’ character preceding either ‘]’ or ‘[’,
3. every ‘\’ character at the end of the link.

Functions inserting links (see Handling Links) properly escape ambiguous characters. You only need to bother about the rules above when inserting directly, or yanking, a URI within square brackets. When in doubt, you may use the function `org-link-escape`, which turns a link string into its escaped form.

Once a link in the buffer is complete, with all brackets present, Org changes the display so that ‘`DESCRIPTION`’ is displayed instead of ‘`[[LINK][DESCRIPTION]]`’ and ‘`LINK`’ is displayed instead of ‘`[[LINK]]`’. Links are highlighted in the `org-link` face, which, by default, is an underlined face.

You can directly edit the visible part of a link. This can be either the *LINK* part, if there is no description, or the *DESCRIPTION* part otherwise. To also edit the invisible *LINK* part, use `C-c C-l` with point on the link (see Handling Links).

If you place point at the beginning or just behind the end of the displayed text and press `BS`, you remove the—invisible—bracket at that location[24]. This makes the link incomplete and the internals are again displayed as plain text. Inserting the missing bracket hides the link internals again. To show the internal structure of all links, use the menu: Org → Hyperlinks → Literal links, customize `org-link-descriptive`, or use ‘`literallinks`’ startup option.

Next: Radio Targets, Previous: Link Format, Up: Hyperlinks [Contents][Index]

4.2 Internal Links

A link that does not look like a URL—i.e., does not start with a known scheme or a file name—refers to the current document. You can follow it with `C-c C-o` when point is on the link, or with a mouse click (see Handling Links).

Org provides several refinements to internal navigation within a document. Most notably, a construct like '`[[#my-custom-id]]`' specifically targets the entry with the '`CUSTOM_ID`' property set to '`my-custom-id`'. Also, an internal link looking like '`[[*Some section]]`' points to a headline with the name '`Some section`'[25].

When the link does not belong to any of the cases above, Org looks for a *dedicated target*: the same string in double angular brackets, like '`<<My Target>>`'.

If no dedicated target exists, the link tries to match the exact name of an element within the buffer. Naming is done, unsurprisingly, with the '`NAME`' keyword, which has to be put in the line before the element it refers to, as in the following example

```
#+NAME: My Target
| a  | table      |
|----+------------|
| of | four cells |
```

Ultimately, if none of the above succeeds, Org searches for a headline that is exactly the link text but may also include a TODO keyword and tags, or initiates a plain text search, according to the value of `org-link-search-must-match-exact-headline`.

Note that you must make sure custom IDs, dedicated targets, and names are unique throughout the document. Org provides a linter to assist you in the process, if needed. See Org Syntax.

During export, internal links are used to mark objects and assign them a number. Marked objects are then referenced by links pointing to them. In particular, links without a description appear as the number assigned to the marked object[26]. In the following excerpt from an Org buffer

```
1. one item
2. <<target>>another item
Here we refer to item [[target]].
```

The last sentence will appear as '`Here we refer to item 2`' when exported.

In non-Org files, the search looks for the words in the link text. In the above example the search would be for '`target`'.

Following a link pushes a mark onto Org's own mark ring. You can return to the previous position with `C-c &`. Using this command several times in direct succession goes back to positions recorded earlier.

4.3 Radio Targets

Org can automatically turn any occurrences of certain target names in normal text into a link. So without explicitly creating a link, the text connects to the target radioing its position. Radio targets are enclosed by triple angular brackets. For example, a target '`<<<My Target>>>`'

causes each occurrence of ‘`my target`’ in normal text to become activated as a link. The Org file is scanned automatically for radio targets only when the file is first loaded into Emacs. To update the target list during editing, press `C-c C-c` with point on or at a target.

4.4 External Links

Org supports links to files, websites, Usenet and email messages, BBDB database entries and links to both IRC conversations and their logs. External links are URL-like locators. They start with a short identifying string followed by a colon. There can be no space after the colon.

Here is the full set of built-in link types:

‘`file`’
File links. File name may be remote, absolute, or relative.

As a special case, “file” prefix may be omitted if the file name is complete, e.g., it starts with ‘`./`’, or ‘`/`’.

‘`attachment`’
Same as file links but for files and folders attached to the current node (see Attachments). Attachment links are intended to behave exactly as file links but for files relative to the attachment directory.

‘`bbdb`’
Link to a BBDB record, with possible regexp completion.

‘`docview`’
Link to a document opened with DocView mode. You may specify a page number.

‘`doi`’
Link to an electronic resource, through its handle.

‘`elisp`’
Execute an Elisp command upon activation.

‘`gnus`’, ‘`rmail`’, ‘`mhe`’
Link to messages or folders from a given Emacs MUA.

‘`help`’
Display documentation of a symbol in ‘`*Help*`’ buffer.

‘`http`’, ‘`https`’
Web links.

‘`id`’
Link to a specific headline by its ID property, in an Org file.

‘`info`’
Link to an Info manual, or to a specific node.

‘`irc`’

Link to an IRC channel.

‘`mailto`’
Link to message composition.

‘`news`’
Usenet links.

‘`shell`’
Execute a shell command upon activation.

For ‘`file:`’ and ‘`id:`’ links, you can additionally specify a line number, or a text search string, separated by ‘`::`’. In Org files, you may link to a headline name, a custom ID, or a code reference instead.

The following table illustrates the link types above, along with their options:

Link Type	Example
http	‘`http://orgmode.org/`’
https	‘`https://orgmode.org/`’
doi	‘`doi:10.1000/182`’
file	‘`file:/home/dominik/images/jupiter.jpg`’
	‘`/home/dominik/images/jupiter.jpg`’ (same as above)
	‘`file:papers/last.pdf`’
	‘`./papers/last.pdf`’ (same as above)
	‘`file:/ssh:me@some.where:papers/last.pdf`’ (remote)
	‘`/ssh:me@some.where:papers/last.pdf`’ (same as above)
	‘`file:sometextfile::NNN`’ (jump to line number)
	‘`file:projects.org`’
	‘`file:projects.org::some words`’ (text search)[27]
	‘`file:projects.org::*task title`’ (headline search)
	‘`file:projects.org::#custom-id`’ (headline search)
attachment	‘`attachment:projects.org`’
	‘`attachment:projects.org::some words`’ (text search)
docview	‘`docview:papers/last.pdf::NNN`’
id	‘`id:B7423F4D-2E8A-471B-8810-C40F074717E9`’

Link Type	Example
	‘`id:B7423F4D-2E8A-471B-8810-C40F074717E9::*task`’ (headline search)
news	‘`news:comp.emacs`’
mailto	‘`mailto:adent@galaxy.net`’
mhe	‘`mhe:folder`’ (folder link)
	‘`mhe:folder#id`’ (message link)
rmail	‘`rmail:folder`’ (folder link)
	‘`rmail:folder#id`’ (message link)
gnus	‘`gnus:group`’ (group link)
	‘`gnus:group#id`’ (article link)
bbdb	‘`bbdb:R.*Stallman`’ (record with regexp)
irc	‘`irc:/irc.com/#emacs/bob`’
help	‘`help:org-store-link`’
info	‘`info:org#External links`’
shell	‘`shell:ls *.org`’
elisp	‘`elisp:(find-file "Elisp.org")`’ (Elisp form to evaluate)
	‘`elisp:org-agenda`’ (interactive Elisp command)

On top of these built-in link types, additional ones are available through the ‘`org-contrib`’ repository (see Installation). For example, these links to VM or Wanderlust messages are available when you load the corresponding libraries from the ‘`org-contrib`’ repository:

‘`vm:folder`’	VM folder link
‘`vm:folder#id`’	VM message link
‘`vm://myself@some.where.org/folder#id`’	VM on remote machine
‘`vm-imap:account:folder`’	VM IMAP folder link
‘`vm-imap:account:folder#id`’	VM IMAP message link
‘`wl:folder`’	Wanderlust folder link
‘`wl:folder#id`’	Wanderlust message link

For information on customizing Org to add new link types, see Adding Hyperlink Types.

A link should be enclosed in double brackets and may contain descriptive text to be displayed instead of the URL (see Link Format), for example:

```
[[https://www.gnu.org/software/emacs/][GNU Emacs]]
```

If the description is a file name or URL that points to an image, HTML export (see HTML Export) inlines the image as a clickable button. If there is no description at all and the link points to an image, that image is inlined into the exported HTML file.

Org also recognizes external links amid normal text and activates them as links. If spaces must be part of the link (for example in '`bbdb:R.*Stallman`'), or if you need to remove ambiguities about the end of the link, enclose the link in square or angular brackets.

4.5 Handling Links

Org provides methods to create a link in the correct syntax, to insert it into an Org file, and to follow the link.

The main function is `org-store-link`, called with `M-x org-store-link`. Because of its importance, we suggest to bind it to a widely available key (see Activation). It stores a link to the current location. The link is stored for later insertion into an Org buffer—see below. The kind of link that is created depends on the current buffer:

Org mode buffers
For Org files, if there is a '`<<target>>`' at point, the link points to the target. If there is a named block (using '`#+name:`') at point, the link points to that name. Otherwise it points to the current headline, which is also the description.

If the headline has a '`CUSTOM_ID`' property, store a link to this custom ID. In addition or alternatively, depending on the value of `org-id-link-to-org-use-id`, create and/or use a globally unique '`ID`' property for the link[28]. So using this command in Org buffers potentially creates two links: a human-readable link from the custom ID, and one that is globally unique and works even if the entry is moved from file to file. The '`ID`' property can be either a UUID (default) or a timestamp, depending on `org-id-method`. Later, when inserting the link, you need to decide which one to use.

When `org-id-link-consider-parent-id` is `t`[29], parent '`ID`' properties are considered. This allows linking to specific targets, named blocks, or headlines (which may not have a globally unique '`ID`' themselves) within the context of a parent headline or file which does.

For example, given this org file:

```
* Parent
:PROPERTIES:
:ID: abc
:END:
** Child 1
** Child 2
```

Storing a link with point at "Child 1" will produce a link '`<id:abc::*Child 1>`', which precisely links to the "Child 1" headline even though it does not have its own ID.

Email/News clients: VM, Rmail, Wanderlust, MH-E, Gnus ¶
Pretty much all Emacs mail clients are supported. The link points to the current article, or, in some Gnus buffers, to the group. The description is constructed according to the variable `org-link-email-description-format`. By default, it refers to the addressee and the subject.

Web browsers: W3M and EWW
Here the link is the current URL, with the page title as the description.

Contacts: BBDB
Links created in a BBDB buffer point to the current entry.

Chat: IRC ¶
For IRC links, if the variable `org-irc-link-to-logs` is non-`nil`, create a '`file`' style link to the relevant point in the logs for the current conversation. Otherwise store an '`irc`' style link to the user/channel/server under the point.

Other files
For any other file, the link points to the file, with a search string (see Search Options in File Links) pointing to the contents of the current line. If there is an active region, the selected words form the basis of the search string. You can write custom Lisp functions to select the search string and perform the search for particular file types (see Custom Searches).

You can also define dedicated links to other files. See Adding Hyperlink Types.

Agenda view
When point is in an agenda view, the created link points to the entry referenced by the current line.

From an Org buffer, the following commands create, navigate or, more generally, act on links.

`C-c C-l` (`org-insert-link`) ¶
Insert a link[30]. This prompts for a link to be inserted into the buffer. You can just type a link, using text for an internal link, or one of the link type prefixes mentioned in the examples above. The link is inserted into the buffer, along with a descriptive text[31]. If some text was selected at this time, it becomes the default description.

Inserting stored links
All links stored during the current session are part of the history for this prompt, so you can access them with `UP` and `DOWN` (or `M-p`, `M-n`).

Completion support
Completion with `TAB` helps you to insert valid link prefixes like '`http`' or '`ftp`', including the prefixes defined through link abbreviations (see Link Abbreviations). If you press `RET` after inserting only the prefix, Org offers specific completion support for some link types[32]. For example, if you type `f i l e RET`—alternative access: `C-u C-c C-l`, see below—Org offers file name completion, and after `b b d b RET` you can complete contact names.

`C-u C-c C-l` ¶

When `C-c C-l` is called with a `C-u` prefix argument, insert a link to a file. You may use file name completion to select the name of the file. The path to the file is inserted relative to the directory of the current Org file, if the linked file is in the current directory or in a sub-directory of it, or if the path is written relative to the current directory using '`../`'. Otherwise an absolute path is used, if possible with '`~/`' for your home directory. You can force an absolute path with two `C-u` prefixes.

`C-c C-l` (with point on existing link) ¶
When point is on an existing link, `C-c C-l` allows you to edit the link and description parts of the link.

`C-c C-o` (`org-open-at-point`) ¶
Open link at point. This launches a web browser for URL (using `browse-url-at-point`), run VM/MH-E/Wanderlust/Rmail/Gnus/BBDB for the corresponding links, and execute the command in a shell link. When point is on an internal link, this command runs the corresponding search. When point is on the tags part of a headline, it creates the corresponding tags view (see Matching tags and properties). If point is on a timestamp, it compiles the agenda for that date. Furthermore, it visits text and remote files in '`file`' links with Emacs and select a suitable application for local non-text files. Classification of files is based on file extension only. See option `org-file-apps`. If you want to override the default application and visit the file with Emacs, use a `C-u` prefix. If you want to avoid opening in Emacs, use a `C-u C-u` prefix.

If point is on a headline, but not on a link, offer all links in the headline and entry text. If you want to setup the frame configuration for following links, customize `org-link-frame-setup`.

`RET` ¶
When `org-return-follows-link` is set, `RET` also follows the link at point.

`mouse-2` or `mouse-1` ¶
On links, `mouse-1` and `mouse-2` opens the link just as `C-c C-o` does.

`mouse-3` ¶
Like `mouse-2`, but force file links to be opened with Emacs, and internal links to be displayed in another window[33].

`C-c %` (`org-mark-ring-push`) ¶
Push the current position onto the Org mark ring, to be able to return easily. Commands following an internal link do this automatically.

`C-c &` (`org-mark-ring-goto`) ¶
Jump back to a recorded position. A position is recorded by the commands following internal links, and by `C-c %`. Using this command several times in direct succession moves through a ring of previously recorded positions.

`C-c C-x C-n` (`org-next-link`) ¶
`C-c C-x C-p` (`org-previous-link`)
Move forward/backward to the next link in the buffer. At the limit of the buffer, the search fails once, and then wraps around. The key bindings for this are really too long; you might want to bind this also to `M-n` and `M-p`.

```
(with-eval-after-load 'org
  (define-key org-mode-map (kbd "M-n") #'org-next-link)
  (define-key org-mode-map (kbd "M-p") #'org-previous-link))
```

4.6 Using Links Outside Org

You can insert and follow links that have Org syntax not only in Org, but in any Emacs buffer. For this, Org provides two functions: `org-insert-link-global` and `org-open-at-point-global`.

You might want to bind them to globally available keys. See Activation for some advice.

4.7 Link Abbreviations

Long URL can be cumbersome to type, and often many similar links are needed in a document. For this you can use link abbreviations. An abbreviated link looks like this

```
[[linkword:tag][description]]
```

where the tag is optional. The *linkword* must be a word, starting with a letter, followed by letters, numbers, '-', and '_'. Abbreviations are resolved according to the information in the variable `org-link-abbrev-alist` that relates the linkwords to replacement text. Here is an example:

```
(setq org-link-abbrev-alist
      '(("bugzilla"        . "https://10.1.2.9/bugzilla/show_bug.cgi?id=")
        ("Nu Html Checker" . "https://validator.w3.org/nu/?doc=%h")
        ("duckduckgo"      . "https://duckduckgo.com/?q=%s")
        ("omap"            . "https://nominatim.openstreetmap.org/search?
q=%s&polygon=1")
        ("ads"             .
"https://ui.adsabs.harvard.edu/search/q=%20author%3A\"%s\"")))
```

If the replacement text contains the string '%s', it is replaced with the tag. Using '%h' instead of '%s' percent-encodes the tag (see the example above, where we need to encode the URL parameter). Using '`%(my-function)`' passes the tag to a custom Lisp function, and replace it by the resulting string.

If the replacement text do not contain any specifier, it is simply appended to the string in order to create the link.

Instead of a string, you may also specify a Lisp function to create the link. Such a function will be called with the tag as the only argument.

With the above setting, you could link to a specific bug with '`[[bugzilla:129]]`', search the web for '`OrgMode`' with '`[[duckduckgo:OrgMode]]`', show the map location of the Free Software Foundation '`[[omap:31 Milk Street, Boston]]`' or of Carsten's office '`[[omap:Science Park 904, Amsterdam, The Netherlands]]`' and find out what the Org author is doing besides Emacs hacking with '`[[ads:Dominik,C]]`'.

If you need special abbreviations just for a single Org buffer, you can define them in the file with

```
#+LINK: bugzilla  https://10.1.2.9/bugzilla/show_bug.cgi?id=
#+LINK: duckduckgo https://duckduckgo.com/?q=%s
#+LINK: "Nu Html Checker" https://validator.w3.org/nu/?doc=%h
```

The abbreviations containing spaces must be quoted.

In-buffer completion (see Completion) can be used after '[' to complete link abbreviations. You may also define a Lisp function that implements special (e.g., completion) support for inserting such a link with `C-c C-l`. Such a function should not accept any arguments, and should return the full link with a prefix. You can set the link completion function like this:

```
(org-link-set-parameter "type" :complete #'some-completion-function)
```

4.8 Search Options in File Links

File links can contain additional information to make Emacs jump to a particular location in the file when following a link. This can be a line number or a search option after a double colon[34]. For example, when the command `org-store-link` creates a link (see Handling Links) to a file, it encodes the words in the current line as a search string that can be used to find this line back later when following the link with `C-c C-o`.

Note that all search options apply for Attachment and ID links in the same way that they apply for File links.

Here is the syntax of the different ways to attach a search to a file link, together with explanations for each:

```
[[file:~/code/main.c::255]]
[[file:~/xx.org::My Target]]
[[file:~/xx.org::*My Target]]
[[file:~/xx.org::#my-custom-id]]
[[file:~/xx.org::/regexp/]]
[[attachment:main.c::255]]
```

'255'
Jump to line 255.

'My Target'
Search for a link target '<<My Target>>', or do a text search for 'my target', similar to the search in internal links, see Internal Links. In HTML export (see HTML Export), such a file link becomes a HTML reference to the corresponding named anchor in the linked file.

'*My Target'
In an Org file, restrict search to headlines.

'#my-custom-id'
Link to a heading with a 'CUSTOM_ID' property

'/REGEXP/'
Do a regular expression search for *REGEXP* (see Regular Expressions). This uses the Emacs command `occur` to list all matches in a separate window. If the target file is in Org mode, `org-occur` is used to create a sparse tree with the matches.

As a degenerate case, a file link with an empty file name can be used to search the current file. For example, '[[file:::find me]]' does a search for 'find me' in the current file, just as '[[find me]]' would.

4.9 Custom Searches

The default mechanism for creating search strings and for doing the actual search related to a file link may not work correctly in all cases. For example, BibTeX database files have many entries like `year="1993"` which would not result in good search strings, because the only unique identification for a BibTeX entry is the citation key.

If you come across such a problem, you can write custom functions to set the right search string for a particular file type, and to do the search for the string in the file. Using `add-hook`, these functions need to be added to the hook variables `org-create-file-search-functions` and `org-execute-file-search-functions`. See the docstring for these variables for more information. Org actually uses this mechanism for BibTeX database files, and you can use the corresponding code as an implementation example. See the file ‘`ol-bibtex.el`’.

5 TODO Items

Org mode does not maintain TODO lists as separate documents[35]. Instead, TODO items are an integral part of the notes file, because TODO items usually come up while taking notes! With Org mode, simply mark any entry in a tree as being a TODO item. In this way, information is not duplicated, and the entire context from which the TODO item emerged is always present.

Of course, this technique for managing TODO items scatters them throughout your notes file. Org mode compensates for this by providing methods to give you an overview of all the things that you have to do.

5.1 Basic TODO Functionality

Any headline becomes a TODO item when it starts with the word ‘`TODO`’, for example:

```
*** TODO Write letter to Sam Fortune
```

The most important commands to work with TODO entries are:

`C-c C-t` (`org-todo`) ¶
Rotate the TODO state of the current item among

```
,-> (unmarked) -> TODO -> DONE --.
'--------------------------------'
```

If TODO keywords have fast access keys (see Fast access to TODO states), prompt for a TODO keyword through the fast selection interface; this is the default behavior when `org-use-fast-todo-selection` is non-`nil`.

The same state changing can also be done “remotely” from the agenda buffer with the `t` command key (see Commands in the Agenda Buffer).

`S-RIGHT` **`S-LEFT`** ¶
Select the following/preceding TODO state, similar to cycling. Useful mostly if more than two TODO states are possible (see Extended Use of TODO Keywords). See also Packages that conflict with Org mode, for a discussion of the interaction with shift-selection. See also the variable `org-treat-S-cursor-todo-selection-as-state-change`.

`C-c / t` (`org-show-todo-tree`) ¶
View TODO items in a *sparse tree* (see Sparse Trees). Folds the entire buffer, but shows all TODO items—with not-DONE state—and the headings hierarchy above them. With a prefix argument, or by using `C-c / T`, search for a specific TODO. You are prompted for the keyword, and you can also give a list of keywords like 'KWD1|KWD2|...' to list entries that match any one of these keywords. With a numeric prefix argument N, show the tree for the Nth keyword in the variable `org-todo-keywords`. With two prefix arguments, find all TODO states, both un-done and done.

`M-x org-agenda t` (`org-todo-list`) ¶
Show the global TODO list. Collects the TODO items (with not-DONE states) from all agenda files (see Agenda Views) into a single buffer. The new buffer is in Org Agenda mode, which provides commands to examine and manipulate the TODO entries from the new buffer (see Commands in the Agenda Buffer). See The global TODO list, for more information.

`S-M-RET` (`org-insert-todo-heading`) ¶
Insert a new TODO entry below the current one.

Changing a TODO state can also trigger tag changes. See the docstring of the option `org-todo-state-tags-triggers` for details.

5.2 Extended Use of TODO Keywords

By default, marked TODO entries have one of only two states: TODO and DONE. Org mode allows you to classify TODO items in more complex ways with *TODO keywords* (stored in `org-todo-keywords`). With special setup, the TODO keyword system can work differently in different files.

Note that *tags* are another way to classify headlines in general and TODO items in particular (see Tags).

5.2.1 TODO keywords as workflow states

You can use TODO keywords to indicate different, possibly *sequential* states in the process of working on an item, for example[36]:

```
(setq org-todo-keywords
      '((sequence "TODO" "FEEDBACK" "VERIFY" "|" "DONE" "DELEGATED")))
```

The vertical bar separates the TODO keywords (states that *need action*) from the DONE states (which need *no further action*). If you do not provide the separator bar, the last state is used as the DONE state.

With this setup, the command `C-c C-t` cycles an entry from 'TODO' to 'FEEDBACK', then to 'VERIFY', and finally to 'DONE' and 'DELEGATED'. You may also use a numeric prefix argument to quickly select a specific state. For example `C-3 C-c C-t` changes the state immediately to 'VERIFY'. Or you can use `S-RIGHT` and `S-LEFT` to go forward and backward through the states. If you define many keywords, you can use in-buffer completion (see Completion) or a special one-key selection scheme (see Fast access to TODO states) to insert these words into the buffer. Changing a TODO state can be logged with a timestamp, see Tracking TODO state changes, for more information.

5.2.2 TODO keywords as types

The second possibility is to use TODO keywords to indicate different *types* of action items. For example, you might want to indicate that items are for “work” or “home”. Or, when you work with several people on a single project, you might want to assign action items directly to persons, by using their names as TODO keywords. This type of functionality is actually much better served by using tags (see Tags), so the TODO implementation is kept just for backward compatibility.

Using TODO types, it would be set up like this:

```
(setq org-todo-keywords '((type "Fred" "Sara" "Lucy" "|" "DONE")))
```

In this case, different keywords do not indicate states, but rather different types. So the normal work flow would be to assign a task to a person, and later to mark it DONE. Org mode supports this style by adapting the workings of the command `C-c C-t`[37]. When used several times in succession, it still cycles through all names, in order to first select the right type for a task. But when you return to the item after some time and execute `C-c C-t` again, it will switch from any name directly to ‘DONE’. Use prefix arguments or completion to quickly select a specific name. You can also review the items of a specific TODO type in a sparse tree by using a numeric prefix to `C-c / t`. For example, to see all things Lucy has to do, you would use `C-3 C-c / t`. To collect Lucy’s items from all agenda files into a single buffer, you would use the numeric prefix argument as well when creating the global TODO list: `C-3 M-x org-agenda t`.

5.2.3 Multiple keyword sets in one file

Sometimes you may want to use different sets of TODO keywords in parallel. For example, you may want to have the basic TODO/DONE, but also a workflow for bug fixing, and a separate state indicating that an item has been canceled—so it is not DONE, but also does not require action. Your setup would then look like this:

```
(setq org-todo-keywords
      '((sequence "TODO" "|" "DONE")
        (sequence "REPORT" "BUG" "KNOWNCAUSE" "|" "FIXED")
        (sequence "|" "CANCELED")))
```

The keywords should all be different, this helps Org mode keep track of which subsequence should be used for a given entry. In this setup, `C-c C-t` only operates within a sub-sequence, so it switches from ‘DONE’ to (nothing) to ‘TODO’, and from ‘FIXED’ to (nothing) to ‘REPORT’. Therefore you need a mechanism to initially select the correct sequence. In addition to typing a keyword or using completion (see Completion), you may also apply the following commands:

`C-u C-u C-c C-t` ¶
`C-S-RIGHT`
`C-S-LEFT`
These keys jump from one TODO sub-sequence to the next. In the above example, `C-u C-u C-c C-t` or `C-S-RIGHT` would jump from ‘TODO’ or ‘DONE’ to ‘REPORT’, and any of the words in the second row to ‘CANCELED’. Note that the `C-S-` key binding conflict with shift-selection (see Packages that conflict with Org mode).

`S-RIGHT` ¶
`S-LEFT`

`S-LEFT` and `S-RIGHT` walk through *all* keywords from all sub-sequences, so for example `S-RIGHT` would switch from 'DONE' to 'REPORT' in the example above. For a discussion of the interaction with shift-selection, see Packages that conflict with Org mode.

5.2.4 Fast access to TODO states

If you would like to quickly change an entry to an arbitrary TODO state instead of cycling through the states, you can set up keys for single-letter access to the states. This is done by adding the selection character after each keyword, in parentheses[38]. For example:

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "|" "DONE(d)")
        (sequence "REPORT(r)" "BUG(b)" "KNOWNCAUSE(k)" "|" "FIXED(f)")
        (sequence "|" "CANCELED(c)")))
```

If you then press `C-c C-t` followed by the selection key, the entry is switched to this state. `SPC` can be used to remove any TODO keyword from an entry.

5.2.5 Setting up keywords for individual files

It can be very useful to use different aspects of the TODO mechanism in different files. For file-local settings, you need to add special lines to the file which set the keywords and interpretation for that file only. For example, to set one of the two examples discussed above, you need one of the following lines, starting in column zero anywhere in the file:

```
#+TODO: TODO FEEDBACK VERIFY | DONE CANCELED
```

You may also write '#+SEQ_TODO' to be explicit about the interpretation, but it means the same as '#+TODO', or

```
#+TYP_TODO: Fred Sara Lucy Mike | DONE
```

A setup for using several sets in parallel would be:

```
#+TODO: TODO(t) | DONE(d)
#+TODO: REPORT(r) BUG(b) KNOWNCAUSE(k) | FIXED(f)
#+TODO: | CANCELED(c)
```

To make sure you are using the correct keyword, type '#+' into the buffer and then use `M-TAB` to complete it (see Completion).

Remember that the keywords after the vertical bar—or the last keyword if no bar is there—must always mean that the item is DONE, although you may use a different word. After changing one of these lines, use `C-c C-c` with point still in the line to make the changes known to Org mode[39].

5.2.6 Faces for TODO keywords

Org mode highlights TODO keywords with special faces: `org-todo` for keywords indicating that an item still has to be acted upon, and `org-done` for keywords indicating that an item is finished. If you are using more than two different states, you might want to use special faces for some of them. This can be done using the variable `org-todo-keyword-faces`. For example:

```
(setq org-todo-keyword-faces
      '(("TODO" . org-warning) ("STARTED" . "yellow")
        ("CANCELED" . (:foreground "blue" :weight bold))))
```

While using a list with face properties as shown for ‘CANCELED’ *should* work, this does not always seem to be the case. If necessary, define a special face and use that. A string is interpreted as a color. The variable `org-faces-easy-properties` determines if that color is interpreted as a foreground or a background color.

5.2.7 TODO dependencies

The structure of Org files—hierarchy and lists—makes it easy to define TODO dependencies. Usually, a parent TODO task should not be marked as done until all TODO subtasks, or children tasks, are marked as done. Sometimes there is a logical sequence to (sub)tasks, so that one subtask cannot be acted upon before all siblings above it have been marked as done. If you customize the variable `org-enforce-todo-dependencies`, Org blocks entries from changing state to DONE while they have TODO children that are not DONE. Furthermore, if an entry has a property ‘ORDERED’, each of its TODO children is blocked until all earlier siblings are marked as done. Here is an example:

```
* TODO Blocked until (two) is done
** DONE one
** TODO two

* Parent
:PROPERTIES:
:ORDERED:  t
:END:
** TODO a
** TODO b, needs to wait for (a)
** TODO c, needs to wait for (a) and (b)
```

You can ensure an entry is never blocked by using the ‘NOBLOCKING’ property (see Properties and Columns):

```
* This entry is never blocked
:PROPERTIES:
:NOBLOCKING: t
:END:
```

`C-c C-x o` (`org-toggle-ordered-property`) ¶
Toggle the ‘ORDERED’ property of the current entry. A property is used for this behavior because this should be local to the current entry, not inherited from entries above like a tag (see Tags). However, if you would like to *track* the value of this property with a tag for better visibility, customize the variable `org-track-ordered-property-with-tag`.

`C-u C-u C-u C-c C-t` ¶
Change TODO state, regardless of any state blocking.

If you set the variable `org-agenda-dim-blocked-tasks`, TODO entries that cannot be marked as done because of unmarked children are shown in a dimmed font or even made invisible in agenda views (see Agenda Views).

You can also block changes of TODO states by using checkboxes (see Checkboxes). If you set the variable `org-enforce-todo-checkbox-dependencies`, an entry that has unchecked checkboxes is blocked from switching to DONE.

If you need more complex dependency structures, for example dependencies between entries in different trees or files, check out the module 'org-depend.el' in the 'org-contrib' repository.

5.3 Progress Logging

To record a timestamp and a note when changing a TODO state, call the command `org-todo` with a prefix argument.

C-u C-c C-t (**org-todo**) ¶
Prompt for a note and record a the time of the TODO state change. The note is inserted as a list item below the headline, but can also be placed into a drawer, see Tracking TODO state changes.

If you want to be more systematic, Org mode can automatically record a timestamp and optionally a note when you mark a TODO item as DONE, or even each time you change the state of a TODO item. This system is highly configurable, settings can be on a per-keyword basis and can be localized to a file or even a subtree. For information on how to clock working time for a task, see Clocking Work Time.

5.3.1 Closing items

The most basic automatic logging is to keep track of *when* a certain TODO item was marked as done. This can be achieved with[40]

```
(setq org-log-done 'time)
```

Then each time you turn an entry from a TODO (not-done) state into any of the DONE states, a line 'CLOSED: [timestamp]' is inserted just after the headline. If you turn the entry back into a TODO item through further state cycling, that line is removed again. If you turn the entry back to a non-TODO state (by pressing **C-c C-t SPC** for example), that line is also removed, unless you set `org-closed-keep-when-no-todo` to non-`nil`. If you want to record a note along with the timestamp, use[41]

```
(setq org-log-done 'note)
```

You are then prompted for a note, and that note is stored below the entry with a 'Closing Note' heading.

5.3.2 Tracking TODO state changes

You might want to automatically keep track of when a state change occurred and maybe take a note about this change. You can either record just a timestamp, or a time-stamped note. These records are inserted after the headline as an itemized list, newest first[42]. When taking a lot of notes, you might want to get the notes out of the way into a drawer (see Drawers). Customize the variable `org-log-into-drawer` to get this behavior—the recommended drawer for this is called 'LOGBOOK'[43]. You can also overrule the setting of this variable for a subtree by setting a 'LOG_INTO_DRAWER' property.

Since it is normally too much to record a note for every state, Org mode expects configuration on a per-keyword basis for this. This is achieved by adding special markers '!' (for a timestamp) or '@' (for a note with timestamp) in parentheses after each keyword. For example, with the setting

```
(setq org-todo-keywords
      '((sequence "TODO(t)" "WAIT(w@/!)" "|" "DONE(d!)" "CANCELED(c@)")))
```

You not only define global TODO keywords and fast access keys, but also request that a time is recorded when the entry is set to 'DONE', and that a note is recorded when switching to 'WAIT' or 'CANCELED'[44]. The setting for 'WAIT' is even more special: the '!' after the slash means that in addition to the note taken when entering the state, a timestamp should be recorded when *leaving* the 'WAIT' state, if and only if the *target* state does not configure logging for entering it. So it has no effect when switching from 'WAIT' to 'DONE', because 'DONE' is configured to record a timestamp only. But when switching from 'WAIT' back to 'TODO', the '/!' in the 'WAIT' setting now triggers a timestamp even though 'TODO' has no logging configured.

You can use the exact same syntax for setting logging preferences local to a buffer:

```
#+TODO: TODO(t) WAIT(w@/!) | DONE(d!) CANCELED(c@)
```

To record a timestamp without a note for TODO keywords configured with '@', just type `C-c C-c` to enter a blank note when prompted.

In order to define logging settings that are local to a subtree or a single item, define a 'LOGGING' property in this entry. Any non-empty 'LOGGING' property resets all logging settings to `nil`. You may then turn on logging for this specific tree using 'STARTUP' keywords like 'lognotedone' or 'logrepeat', as well as adding state specific settings like 'TODO(!)'. For example:

```
* TODO Log each state with only a time
  :PROPERTIES:
  :LOGGING: TODO(!) WAIT(!) DONE(!) CANCELED(!)
  :END:
* TODO Only log when switching to WAIT, and when repeating
  :PROPERTIES:
  :LOGGING: WAIT(@) logrepeat
  :END:
* TODO No logging at all
  :PROPERTIES:
  :LOGGING: nil
  :END:
```

5.3.3 Tracking your habits

Org has the ability to track the consistency of a special category of TODO, called "habits." To use habits, you have to enable the `habit` module by customizing the variable `org-modules`.

A habit has the following properties:

1. The habit is a TODO item, with a TODO keyword representing an open state.
2. The property 'STYLE' is set to the value 'habit' (see Properties and Columns).

3. The TODO has a scheduled date, usually with a '`.+`' style repeat interval. A '++' style may be appropriate for habits with time constraints, e.g., must be done on specific days of week ('`++1w`'), or a '+' style for an unusual habit that can have a backlog, e.g., weekly reports. See Repeated tasks for more details about repeat intervals.
4. The TODO may also have minimum and maximum ranges specified by using the syntax '`.+2d/3d`', which says that you want to do the task at least every three days, but at most every two days.
5. State logging for the DONE state is enabled (see Tracking TODO state changes), in order for historical data to be represented in the consistency graph. If it is not enabled it is not an error, but the consistency graphs are largely meaningless.

To give you an idea of what the above rules look like in action, here's an actual habit with some history:

```
** TODO Shave
   SCHEDULED: <2009-10-17 Sat .+2d/4d>
   :PROPERTIES:
   :STYLE:    habit
   :LAST_REPEAT: [2009-10-19 Mon 00:36]
   :END:
   - State "DONE"       from "TODO"       [2009-10-15 Thu]
   - State "DONE"       from "TODO"       [2009-10-12 Mon]
   - State "DONE"       from "TODO"       [2009-10-10 Sat]
   - State "DONE"       from "TODO"       [2009-10-04 Sun]
   - State "DONE"       from "TODO"       [2009-10-02 Fri]
   - State "DONE"       from "TODO"       [2009-09-29 Tue]
   - State "DONE"       from "TODO"       [2009-09-25 Fri]
   - State "DONE"       from "TODO"       [2009-09-19 Sat]
   - State "DONE"       from "TODO"       [2009-09-16 Wed]
   - State "DONE"       from "TODO"       [2009-09-12 Sat]
```

What this habit says is: I want to shave at most every 2 days—given by the '`SCHEDULED`' date and repeat interval—and at least every 4 days. If today is the 15th, then the habit first appears in the agenda (see Agenda Views) on Oct 17, after the minimum of 2 days has elapsed, and will appear overdue on Oct 19, after four days have elapsed.

What's really useful about habits is that they are displayed along with a consistency graph, to show how consistent you've been at getting that task done in the past. This graph shows every day that the task was done over the past three weeks, with colors for each day. The colors used are:

Blue
If the task was not to be done yet on that day.

Green
If the task could have been done on that day.

Yellow
If the task was going to be overdue the next day.

Red
If the task was overdue on that day.

In addition to coloring each day, the day is also marked with an asterisk if the task was actually done that day, and an exclamation mark to show where the current day falls in the graph.

There are several configuration variables that can be used to change the way habits are displayed in the agenda.

`org-habit-graph-column` ¶
The buffer column at which the consistency graph should be drawn. This overwrites any text in that column, so it is a good idea to keep your habits' titles brief and to the point.

`org-habit-preceding-days` ¶
The amount of history, in days before today, to appear in consistency graphs.

`org-habit-following-days` ¶
The number of days after today that appear in consistency graphs.

`org-habit-show-habits-only-for-today` ¶
If non-`nil`, only show habits in today's agenda view. The default value is `t`. Pressing `C-u K` in the agenda toggles this variable.

Lastly, pressing `K` in the agenda buffer causes habits to temporarily be disabled and do not appear at all. Press `K` again to bring them back. They are also subject to tag filtering, if you have habits which should only be done in certain contexts, for example.

5.4 Priorities

If you use Org mode extensively, you may end up with enough TODO items that it starts to make sense to prioritize them. Prioritizing can be done by placing a *priority cookie* into the headline of a TODO item right after the TODO keyword, like this:

```
*** TODO [#A] Write letter to Sam Fortune
```

By default, Org mode supports three priorities: 'A', 'B', and 'C'. 'A' is the highest priority. An entry without a cookie is treated as equivalent if it had priority 'B'. Priorities make a difference only for sorting in the agenda (see Weekly/daily agenda). Outside the agenda, they have no inherent meaning to Org mode. The cookies are displayed with the face defined by the variable `org-priority-faces`, which can be customized.

You can also use numeric values for priorities, such as

```
*** TODO [#1] Write letter to Sam Fortune
```

When using numeric priorities, you need to set `org-priority-highest`, `org-priority-lowest` and `org-priority-default` to integers, which must all be strictly inferior to 65.

Priorities can be attached to any outline node; they do not need to be TODO items.

`C-c ,` (**`org-priority`**) ¶
Set the priority of the current headline. The command prompts for a priority character 'A', 'B' or 'C'. When you press `SPC` instead, the priority cookie, if one is set, is removed from the headline. The priorities can also be changed "remotely" from the agenda buffer with the `,` command (see Commands in the Agenda Buffer).

`S-UP` (`org-priority-up`) ¶
`S-DOWN` (`org-priority-down`)
Increase/decrease the priority of the current headline[45]. Note that these keys are also used to modify timestamps (see Creating Timestamps). See also Packages that conflict with Org mode, for a discussion of the interaction with shift-selection.

You can change the range of allowed priorities by setting the variables `org-priority-highest`, `org-priority-lowest`, and `org-priority-default`. For an individual buffer, you may set these values (highest, lowest, default) like this (please make sure that the highest priority is earlier in the alphabet than the lowest priority):

```
#+PRIORITIES: A C B
```

Or, using numeric values:

```
#+PRIORITIES: 1 10 5
```

5.5 Breaking Down Tasks into Subtasks

It is often advisable to break down large tasks into smaller, manageable subtasks. You can do this by creating an outline tree below a TODO item, with detailed subtasks on the tree[46]. To keep an overview of the fraction of subtasks that have already been marked as done, insert either '`[/]`' or '`[%]`' anywhere in the headline. These cookies are updated each time the TODO status of a child changes, or when pressing `C-c C-c` on the cookie. For example:

```
* Organize Party [33%]
** TODO Call people [1/2]
*** TODO Peter
*** DONE Sarah
** TODO Buy food
** DONE Talk to neighbor
```

If a heading has both checkboxes and TODO children below it, the meaning of the statistics cookie become ambiguous. Set the property '`COOKIE_DATA`' to either '`checkbox`' or '`todo`' to resolve this issue.

If you would like to have the statistics cookie count any TODO entries in the subtree (not just direct children), configure the variable `org-hierarchical-todo-statistics`. To do this for a single subtree, include the word '`recursive`' into the value of the '`COOKIE_DATA`' property.

```
* Parent capturing statistics [2/20]
  :PROPERTIES:
  :COOKIE_DATA: todo recursive
  :END:
```

If you would like a TODO entry to automatically change to DONE when all children are done, you can use the following setup:

```
(defun org-summary-todo (n-done n-not-done)
  "Switch entry to DONE when all subentries are done, to TODO otherwise."
  (let (org-log-done org-todo-log-states)   ; turn off logging
    (org-todo (if (= n-not-done 0) "DONE" "TODO"))))

(add-hook 'org-after-todo-statistics-hook #'org-summary-todo)
```

Another possibility is the use of checkboxes to identify (a hierarchy of) a large number of subtasks (see Checkboxes).

5.6 Checkboxes

Every item in a plain list[47] (see Plain Lists) can be made into a checkbox by starting it with the string '`[ ]`'. This feature is similar to TODO items (see TODO Items), but is more lightweight. Checkboxes are not included into the global TODO list, so they are often great to split a task into a number of simple steps. Or you can use them in a shopping list.

Here is an example of a checkbox list.

```
* TODO Organize party [2/4]
  - [-] call people [1/3]
    - [ ] Peter
    - [X] Sarah
    - [ ] Sam
  - [X] order food
  - [ ] think about what music to play
  - [X] talk to the neighbors
```

The '`[2/4]`' and '`[1/3]`' in the first and second line are cookies indicating how many checkboxes present in this entry have been checked off, and the total number of checkboxes present. This can give you an idea on how many checkboxes remain, even without opening a folded entry. The cookies can be placed into a headline or into (the first line of) a plain list item. Each cookie covers checkboxes of direct children structurally below the headline/item on which the cookie appears[48]. You have to insert the cookie yourself by typing either '`[/]`' or '`[%]`'. With '`[/]`' you get an '`n out of m`' result, as in the examples above. With '`[%]`' you get information about the percentage of checkboxes checked (in the above example, this would be '`[50%]`' and '`[33%]`', respectively). In a headline, a cookie can count either checkboxes below the heading or TODO states of children, and it displays whatever was changed last. Set the property '`COOKIE_DATA`' to either '`checkbox`' or '`todo`' to resolve this issue.

If the current outline node has an '`ORDERED`' property, checkboxes must be checked off in sequence, and an error is thrown if you try to check off a box while there are unchecked boxes above it.

A checkbox can be in one of the three states:

1. not checked '`[ ]`'
2. partially checked '`[-]`'
3. checked '`[X]`'

Checkboxes work hierarchically, so if a checkbox item has children that are checkboxes, toggling one of the children checkboxes makes the parent checkbox reflect if none, some, or all of the children are checked.

If all child checkboxes are not checked, the parent checkbox is also not checked.

```
- [ ] call people
  - [ ] Peter
  - [ ] Sarah
```

If some but not all child checkboxes are checked, the parent checkbox is partially checked.

```
- [-] call people
  - [X] Peter
  - [ ] Sarah
```

If all child checkboxes are checked, the parent checkbox is also checked.

```
- [X] call people
  - [X] Peter
  - [X] Sarah
```

The following commands work with checkboxes:

C-c C-c (**org-toggle-checkbox**) ¶
Toggle checkbox status or—with prefix argument—checkbox presence at point. With a single prefix argument, add an empty checkbox or remove the current one[49]. With a double prefix argument, set it to '[-]', which is considered to be an intermediate state.

C-c C-x C-b (**org-toggle-checkbox**) ¶
Toggle checkbox status or—with prefix argument—checkbox presence at point. With double prefix argument, set it to '[-]', which is considered to be an intermediate state.

- If there is an active region, toggle the first checkbox in the region and set all remaining boxes to the same status as the first. With a prefix argument, add or remove the checkbox for all items in the region.
- If point is in a headline, toggle checkboxes in the region between this headline and the next—so *not* the entire subtree.
- If there is no active region, just toggle the checkbox at point.

C-c C-x C-r (**org-toggle-radio-button**) ¶
Toggle checkbox status by using the checkbox of the item at point as a radio button: when the checkbox is turned on, all other checkboxes on the same level will be turned off. With a universal prefix argument, toggle the presence of the checkbox. With a double prefix argument, set it to '[-]'.

`C-c C-c` can be told to consider checkboxes as radio buttons by setting '#+ATTR_ORG: :radio t' right before the list or by calling `M-x org-list-checkbox-radio-mode` to activate this minor mode.

M-S-RET (**org-insert-todo-heading**) ¶
Insert a new item with a checkbox. This works only if point is already in a plain list item (see Plain Lists).

C-c C-x o (**org-toggle-ordered-property**) ¶
Toggle the 'ORDERED' property of the entry, to toggle if checkboxes must be checked off in sequence. A property is used for this behavior because this should be local to the current entry, not inherited like a tag. However, if you would like to *track* the value of this property with a tag for better visibility, customize `org-track-ordered-property-with-tag`.

C-c # (**org-update-statistics-cookies**) ¶

Update the statistics cookie in the current outline entry. When called with a `C-u` prefix, update the entire file. Checkbox statistic cookies are updated automatically if you toggle checkboxes with `C-c C-c` and make new ones with `M-S-RET`. TODO statistics cookies update when changing TODO states. If you delete boxes/entries or add/change them by hand, use this command to get things back into sync.

Next: Properties and Columns, Previous: TODO Items, Up: The Org Manual [Contents][Index]

6 Tags

An excellent way to implement labels and contexts for cross-correlating information is to assign *tags* to headlines. Org mode has extensive support for tags.

Every headline can contain a list of tags; they occur at the end of the headline. Tags are normal words containing letters, numbers, '_', and '@'. Tags must be preceded and followed by a single colon, e.g., '`:work:`'. Several tags can be specified, as in '`:work:urgent:`'. Tags by default are in bold face with the same color as the headline. You may specify special faces for specific tags using the variable `org-tag-faces`, in much the same way as you can for TODO keywords (see Faces for TODO keywords).

6.1 Tag Inheritance

Tags make use of the hierarchical structure of outline trees. If a heading has a certain tag, all subheadings inherit the tag as well. For example, in the list

```
* Meeting with the French group      :work:
** Summary by Frank                  :boss:notes:
*** TODO Prepare slides for him      :action:
```

the final heading has the tags '`work`', '`boss`', '`notes`', and '`action`' even though the final heading is not explicitly marked with those tags. You can also set tags that all entries in a file should inherit just as if these tags were defined in a hypothetical level zero that surrounds the entire file. Use a line like this[50]

```
#+FILETAGS: :Peter:Boss:Secret:
```

To limit tag inheritance to specific tags, or to turn it off entirely, use the variables `org-use-tag-inheritance` and `org-tags-exclude-from-inheritance`.

Tag inheritance is relevant when the agenda search tries to match a tag, either in the `tags` or `tags-todo` agenda types. In other agenda types, `org-use-tag-inheritance` has no effect. Still, you may want to have your tags correctly set in the agenda, so that tag filtering works fine, with inherited tags. Set `org-agenda-use-tag-inheritance` to control this: the default value includes all agenda types, but setting this to `nil` can really speed up agenda generation.

6.2 Setting Tags

Tags can simply be typed into the buffer at the end of a headline. After a colon, `M-TAB` offers completion on tags. There is also a special command for inserting tags:

`C-c C-q` (**`org-set-tags-command`**) ¶

Enter new tags for the current headline. Org mode either offers completion or a special single-key interface for setting tags, see below. After pressing `RET`, the tags are inserted and aligned to `org-tags-column`. When called with a `C-u` prefix, all tags in the current buffer are aligned to that column, just to make things look nice. Tags are automatically realigned after promotion, demotion, and TODO state changes (see Basic TODO Functionality).

`C-c C-c` (**`org-set-tags-command`**) ¶
When point is in a headline, this does the same as `C-c C-q`.

Org supports tag insertion based on a *list of tags*. By default this list is constructed dynamically, containing all tags currently used in the buffer[51]. You may also globally specify a hard list of tags with the variable `org-tag-alist`. Finally you can set the default tags for a given file using the 'TAGS' keyword, like

```
#+TAGS: @work @home @tennisclub
#+TAGS: laptop car pc sailboat
```

If you have globally defined your preferred set of tags using the variable `org-tag-alist`, but would like to use a dynamic tag list in a specific file, add an empty 'TAGS' keyword to that file:

```
#+TAGS:
```

If you have a preferred set of tags that you would like to use in every file, in addition to those defined on a per-file basis by 'TAGS' keyword, then you may specify a list of tags with the variable `org-tag-persistent-alist`. You may turn this off on a per-file basis by adding a 'STARTUP' keyword to that file:

```
#+STARTUP: noptag
```

By default Org mode uses the standard minibuffer completion facilities for entering tags. However, it also implements another, quicker, tag selection method called *fast tag selection*. This allows you to select and deselect tags with just a single key press. For this to work well you should assign unique letters to most of your commonly used tags. You can do this globally by configuring the variable `org-tag-alist` in your Emacs init file. For example, you may find the need to tag many items in different files with '@home'. In this case you can set something like:

```
(setq org-tag-alist '(("@work" . ?w) ("@home" . ?h) ("laptop" . ?l)))
```

If the tag is only relevant to the file you are working on, then you can instead set the 'TAGS' keyword as:

```
#+TAGS: @work(w)  @home(h)  @tennisclub(t)  laptop(l)  pc(p)
```

The tags interface shows the available tags in a splash window. If you want to start a new line after a specific tag, insert '\n' into the tag list

```
#+TAGS: @work(w) @home(h) @tennisclub(t) \n laptop(l) pc(p)
```

or write them in two lines:

```
#+TAGS: @work(w)  @home(h)  @tennisclub(t)
#+TAGS: laptop(l)  pc(p)
```

You can also group together tags that are mutually exclusive by using braces, as in:

```
#+TAGS: { @work(w)  @home(h)  @tennisclub(t) }  laptop(l)  pc(p)
```

you indicate that at most one of '`@work`', '`@home`', and '`@tennisclub`' should be selected. Multiple such groups are allowed.

Do not forget to press `C-c C-c` with point in one of these lines to activate any changes.

To set these mutually exclusive groups in the variable `org-tags-alist`, you must use the dummy tags `:startgroup` and `:endgroup` instead of the braces. Similarly, you can use `:newline` to indicate a line break. The previous example would be set globally by the following configuration:

```
(setq org-tag-alist '((:startgroup . nil)
                      ("@work" . ?w) ("@home" . ?h)
                      ("@tennisclub" . ?t)
                      (:endgroup . nil)
                      ("laptop" . ?l) ("pc" . ?p)))
```

If at least one tag has a selection key then pressing `C-c C-c` automatically presents you with a special interface, listing inherited tags, the tags of the current headline, and a list of all valid tags with corresponding keys[52].

Pressing keys assigned to tags adds or removes them from the list of tags in the current line. Selecting a tag in a group of mutually exclusive tags turns off any other tag from that group.

In this interface, you can also use the following special keys:

TAB ¶
Enter a tag in the minibuffer, even if the tag is not in the predefined list. You can complete on all tags present in the buffer and globally pre-defined tags from `org-tag-alist` and `org-tag-persistent-alist`. You can also add several tags: just separate them with a comma.

SPC ¶
Clear all tags for this line.

RET ¶
Accept the modified set.

C-g ¶
Abort without installing changes.

q ¶
If `q` is not assigned to a tag, it aborts like `C-g`.

! ¶
Turn off groups of mutually exclusive tags. Use this to (as an exception) assign several tags from such a group.

C-c ¶
Toggle auto-exit after the next change (see below). If you are using expert mode, the first `C-c` displays the selection window.

This method lets you assign tags to a headline with very few keys. With the above setup, you could clear the current tags and set '@home', 'laptop' and 'pc' tags with just the following keys: `C-c C-c SPC h l p RET`. Switching from '@home' to '@work' would be done with `C-c C-c w RET` or alternatively with `C-c C-c C-c w`. Adding the non-predefined tag 'sarah' could be done with `C-c C-c TAB s a r a h RET`.

If you find that most of the time you need only a single key press to modify your list of tags, set the variable `org-fast-tag-selection-single-key`. Then you no longer have to press `RET` to exit fast tag selection—it exits after the first change. If you then occasionally need more keys, press `C-c` to turn off auto-exit for the current tag selection process (in effect: start selection with `C-c C-c C-c` instead of `C-c C-c`). If you set the variable to the value `expert`, the special window is not even shown for single-key tag selection, it comes up only when you press an extra `C-c`.

The number of tags displayed in the fast tag selection interface is limited by `org-fast-tag-selection-maximum-tags` to avoid running out of keyboard keys. You can customize this variable.

6.3 Tag Hierarchy

Tags can be defined in hierarchies. A tag can be defined as a *group tag* for a set of other tags. The group tag can be seen as the "broader term" for its set of tags. Defining multiple group tags and nesting them creates a tag hierarchy.

One use-case is to create a taxonomy of terms (tags) that can be used to classify nodes in a document or set of documents.

When you search for a group tag, it return matches for all members in the group and its subgroups. In an agenda view, filtering by a group tag displays or hide headlines tagged with at least one of the members of the group or any of its subgroups. This makes tag searches and filters even more flexible.

You can set group tags by using brackets and inserting a colon between the group tag and its related tags—beware that all whitespaces are mandatory so that Org can parse this line correctly:

```
#+TAGS: [ GTD : Control Persp ]
```

In this example, 'GTD' is the group tag and it is related to two other tags: 'Control', 'Persp'. Defining 'Control' and 'Persp' as group tags creates a hierarchy of tags:

```
#+TAGS: [ Control : Context Task ]
#+TAGS: [ Persp : Vision Goal AOF Project ]
```

That can conceptually be seen as a hierarchy of tags:

- ‘GTD’
  - ‘Persp’
    - ‘Vision’
    - ‘Goal’
    - ‘AOF’
    - ‘Project’
  - ‘Control’
    - ‘Context’
    - ‘Task’

You can use the `:startgrouptag`, `:grouptags` and `:endgrouptag` keyword directly when setting `org-tag-alist` directly:

```
(setq org-tag-alist '((:startgrouptag)
                      ("GTD")
                      (:grouptags)
                      ("Control")
                      ("Persp")
                      (:endgrouptag)
                      (:startgrouptag)
                      ("Control")
                      (:grouptags)
                      ("Context")
                      ("Task")
                      (:endgrouptag)))
```

The tags in a group can be mutually exclusive if using the same group syntax as is used for grouping mutually exclusive tags together; using curly brackets.

```
#+TAGS: { Context : @Home @Work @Call }
```

When setting `org-tag-alist` you can use `:startgroup` and `:endgroup` instead of `:startgrouptag` and `:endgrouptag` to make the tags mutually exclusive.

Furthermore, the members of a group tag can also be regular expressions, creating the possibility of a more dynamic and rule-based tag structure (see Regular Expressions). The regular expressions in the group must be specified within curly brackets. Here is an expanded example:

```
#+TAGS: [ Vision : {V@.+} ]
#+TAGS: [ Goal : {G@.+} ]
#+TAGS: [ AOF : {AOF@.+} ]
#+TAGS: [ Project : {P@.+} ]
```

Searching for the tag ‘`Project`’ now lists all tags also including regular expression matches for ‘`P@.+`’, and similarly for tag searches on ‘`Vision`’, ‘`Goal`’ and ‘`AOF`’. For example, this would work well for a project tagged with a common project-identifier, e.g., ‘`P@2014_OrgTags`’.

If you want to ignore group tags temporarily, toggle group tags support with `org-toggle-tags-groups`, bound to `C-c C-x q`. If you want to disable tag groups completely, set `org-group-tags` to `nil`.

6.4 Tag Searches

Once a system of tags has been set up, it can be used to collect related information into special lists.

`C-c / m` or `C-c \` (`org-match-sparse-tree`) ¶
Create a sparse tree with all headlines matching a tags search. With a `C-u` prefix argument, ignore headlines that are not a TODO line.

`M-x org-agenda m` (`org-tags-view`) ¶
Create a global list of tag matches from all agenda files. See Matching tags and properties.

`M-x org-agenda M` (`org-tags-view`) ¶
Create a global list of tag matches from all agenda files, but check only TODO items.

These commands all prompt for a match string which allows basic Boolean logic like '`+boss+urgent-project1`', to find entries with tags '`boss`' and '`urgent`', but not '`project1`', or '`Kathy|Sally`' to find entries which are tagged, like '`Kathy`' or '`Sally`'. The full syntax of the search string is rich and allows also matching against TODO keywords, entry levels and properties. For a complete description with many examples, see Matching tags and properties.

7 Properties and Columns

A property is a key-value pair associated with an entry. Properties can be set so they are associated with a single entry, with every entry in a tree, or with the whole buffer.

There are two main applications for properties in Org mode. First, properties are like tags, but with a value. Imagine maintaining a file where you document bugs and plan releases for a piece of software. Instead of using tags like '`release_1`', '`release_2`', you can use a property, say '`Release`', that in different subtrees has different values, such as '`1.0`' or '`2.0`'. Second, you can use properties to implement (very basic) database capabilities in an Org buffer. Imagine keeping track of your music CDs, where properties could be things such as the album, artist, date of release, number of tracks, and so on.

Properties can be conveniently edited and viewed in column view (see Column View).

7.1 Property Syntax

Properties are key–value pairs. When they are associated with a single entry or with a tree they need to be inserted into a special drawer (see Drawers) with the name '`PROPERTIES`', which has to be located right below a headline, and its planning line (see Deadlines and Scheduling) when applicable. Each property is specified on a single line, with the key—surrounded by colons—first, and the value after it. Keys are case-insensitive. Here is an example:

```
* CD collection
** Classic
*** Goldberg Variations
    :PROPERTIES:
    :Title:     Goldberg Variations
    :Composer:  J.S. Bach
    :Artist:    Glenn Gould
    :Publisher: Deutsche Grammophon
    :NDisks:    1
    :END:
```

Depending on the value of `org-use-property-inheritance`, a property set this way is associated either with a single entry, or with the subtree defined by the entry, see Property Inheritance.

You may define the allowed values for a particular property ‘`Xyz`’ by setting a property ‘`Xyz_ALL`’. This special property is *inherited*, so if you set it in a level 1 entry, it applies to the entire tree. When allowed values are defined, setting the corresponding property becomes easier and is less prone to typing errors. For the example with the CD collection, we can pre-define publishers and the number of disks in a box like this:

```
* CD collection
  :PROPERTIES:
  :NDisks_ALL:  1 2 3 4
  :Publisher_ALL: "Deutsche Grammophon" Philips EMI
  :END:
```

Properties can be inserted at the buffer level. That means they apply before the first headline and can be inherited by all entries in a file. Property blocks defined before the first headline must be at the top of the buffer with only comments above them.

Properties can also be defined using lines like:

```
#+PROPERTY: NDisks_ALL 1 2 3 4
```

If you want to add to the value of an existing property, append a ‘+’ to the property name. The following results in the property ‘`var`’ having the value ‘`foo=1 bar=2`’.

```
#+PROPERTY: var  foo=1
#+PROPERTY: var+ bar=2
```

It is also possible to add to the values of inherited properties. The following results in the ‘`Genres`’ property having the value ‘`Classic Baroque`’ under the ‘`Goldberg Variations`’ subtree.

```
* CD collection
** Classic
    :PROPERTIES:
    :Genres: Classic
    :END:
*** Goldberg Variations
    :PROPERTIES:
    :Title:     Goldberg Variations
    :Composer:  J.S. Bach
    :Artist:    Glenn Gould
    :Publisher: Deutsche Grammophon
    :NDisks:    1
    :Genres+:   Baroque
    :END:
```

Note that a property can only have one entry per drawer.

Property values set with the global variable `org-global-properties` can be inherited by all entries in all Org files.

The following commands help to work with properties:

M-TAB (**pcomplete**)¶
After an initial colon in a line, complete property keys. All keys used in the current file are offered as possible completions.

C-c C-x p (**org-set-property**)¶
Set a property. This prompts for a property name and a value. If necessary, the property drawer is created as well.

C-u M-x org-insert-drawer¶
Insert a property drawer into the current entry. The drawer is inserted early in the entry, but after the lines with planning information like deadlines. If before first headline the drawer is inserted at the top of the drawer after any potential comments.

C-c C-c (**org-property-action**)¶
With point in a property drawer, this executes property commands.

C-c C-c s (**org-set-property**)¶
Set a property in the current entry. Both the property and the value can be inserted using completion.

S-RIGHT (**org-property-next-allowed-value**)¶
S-LEFT (**org-property-previous-allowed-value**)
Switch property at point to the next/previous allowed value.

C-c C-c d (**org-delete-property**)¶
Remove a property from the current entry.

C-c C-c D (**org-delete-property-globally**)¶
Globally remove a property, from all entries in the current file.

C-c C-c c (**org-compute-property-at-point**)¶

Compute the property at point, using the operator and scope from the nearest column format definition.

7.2 Special Properties

Special properties provide an alternative access method to Org mode features, like the TODO state or the priority of an entry, discussed in the previous chapters. This interface exists so that you can include these states in a column view (see Column View), or to use them in queries. The following property names are special and should not be used as keys in the properties drawer:

'ALLTAGS'	All tags, including inherited ones.
'BLOCKED'	`t` if task is currently blocked by children or siblings.
'CATEGORY'	The category of an entry.
'CLOCKSUM'	The sum of CLOCK intervals in the subtree. `org-clock-sum`
	must be run first to compute the values in the current buffer.
'CLOCKSUM_T'	The sum of CLOCK intervals in the subtree for today.
	`org-clock-sum-today` must be run first to compute the
	values in the current buffer.
'CLOSED'	When was this entry closed?
'DEADLINE'	The deadline timestamp.
'FILE'	The filename the entry is located in.
'ITEM'	The headline of the entry.
'PRIORITY'	The priority of the entry, a string with a single letter.
'SCHEDULED'	The scheduling timestamp.
'TAGS'	The tags defined directly in the headline.
'TIMESTAMP'	The first active keyword-less timestamp in the entry.[53]
'TIMESTAMP_IA'	The first inactive keyword-less timestamp in the entry.
'TODO'	The TODO keyword of the entry.

7.3 Property Searches

To create sparse trees and special lists with selection based on properties, the same commands are used as for tag searches (see Tag Searches).

`C-c / m` or `C-c \` (`org-match-sparse-tree`) ¶
Create a sparse tree with all matching entries. With a `C-u` prefix argument, ignore headlines that are not a TODO line.

`M-x org-agenda m` (`org-tags-view`) ¶
Create a global list of tag/property matches from all agenda files.

`M-x org-agenda M` (`org-tags-view`) ¶
Create a global list of tag matches from all agenda files, but check only TODO items.

The syntax for the search string is described in Matching tags and properties.

There is also a special command for creating sparse trees based on a single property:

`C-c / p` ¶
Create a sparse tree based on the value of a property. This first prompts for the name of a property, and then for a value. A sparse tree is created with all entries that define this property with the given value. If you enclose the value in curly braces, it is interpreted as a regular expression and matched against the property values (see Regular Expressions).

7.4 Property Inheritance

The outline structure of Org documents lends itself to an inheritance model of properties: if the parent in a tree has a certain property, the children can inherit this property. Org mode does not turn this on by default, because it can slow down property searches significantly and is often not needed. However, if you find inheritance useful, you can turn it on by setting the variable `org-use-property-inheritance`. It may be set to `t` to make all properties inherited from the parent, to a list of properties that should be inherited, or to a regular expression that matches inherited properties. If a property has the value `nil`, this is interpreted as an explicit un-define of the property, so that inheritance search stops at this value and returns `nil`.

Org mode has a few properties for which inheritance is hard-coded, at least for the special applications for which they are used:

`COLUMNS` ¶
The ‘`COLUMNS`’ property defines the format of column view (see Column View). It is inherited in the sense that the level where a ‘`COLUMNS`’ property is defined is used as the starting point for a column view table, independently of the location in the subtree from where columns view is turned on.

`CATEGORY` ¶
For agenda view, a category set through a ‘`CATEGORY`’ property applies to the entire subtree.

`ARCHIVE` ¶
For archiving, the ‘`ARCHIVE`’ property may define the archive location for the entire subtree (see Moving a tree to an archive file).

`LOGGING` ¶
The ‘`LOGGING`’ property may define logging settings for an entry or a subtree (see Tracking TODO state changes).

7.5 Column View

A great way to view and edit properties in an outline tree is *column view*. In column view, each outline node is turned into a table row. Columns in this table provide access to properties of the entries. Org mode implements columns by overlaying a tabular structure over the headline of each item. While the headlines have been turned into a table row, you can still change the visibility of the outline tree. For example, you get a compact table by switching to “contents” view—`S-TAB` `S-TAB`, or simply `c` while column view is active—but you can still open, read, and edit the entry below each headline. Or, you can switch to column view after executing a sparse tree command and in this way get a table only for the selected items. Column view also works in agenda buffers (see Agenda Views) where queries have collected selected items, possibly from a number of files.

7.5.1 Defining columns

Setting up a column view first requires defining the columns. This is done by defining a column format line.

7.5.1.1 Scope of column definitions

To specify a format that only applies to a specific tree, add a ‘COLUMNS’ property to the top node of that tree, for example:

```
** Top node for columns view
   :PROPERTIES:
   :COLUMNS: %25ITEM %TAGS %PRIORITY %TODO
   :END:
```

A ‘COLUMNS’ property within a property drawer before first headline will apply to the entire file. As an addition to property drawers, keywords can also be defined for an entire file using a line like:

```
#+COLUMNS: %25ITEM %TAGS %PRIORITY %TODO
```

If a ‘COLUMNS’ property is present in an entry, it defines columns for the entry itself, and for the entire subtree below it. Since the column definition is part of the hierarchical structure of the document, you can define columns on level 1 that are general enough for all sublevels, and more specific columns further down, when you edit a deeper part of the tree.

7.5.1.2 Column attributes

A column definition sets the attributes of a column. The general definition looks like this:

```
%[WIDTH]PROPERTY[(TITLE)][{SUMMARY-TYPE}]
```

Except for the percent sign and the property name, all items are optional. The individual parts have the following meaning:

WIDTH
An integer specifying the width of the column in characters. If omitted, the width is determined automatically.

PROPERTY
The property that should be edited in this column. Special properties representing meta data are allowed here as well (see Special Properties).

TITLE
The header text for the column. If omitted, the property name is used.

SUMMARY-TYPE
The summary type. If specified, the column values for parent nodes are computed from the children[54].

Supported summary types are:

'+'	Sum numbers in this column.
'`+;%.1f`'	Like '+', but format result with '`%.1f`'.
'$'	Currency, short for '`+;%.2f`'.
'`min`'	Smallest number in column.
'`max`'	Largest number.
'`mean`'	Arithmetic mean of numbers.
'X'	Checkbox status, '`[X]`' if all children are '`[X]`'.
'X/'	Checkbox status, '`[n/m]`'.
'X%'	Checkbox status, '`[n%]`'.
':'	Sum times, HH:MM, plain numbers are minutes.
'`:min`'	Smallest time value in column.
'`:max`'	Largest time value.
'`:mean`'	Arithmetic mean of time values.
'`@min`'	Minimum age[55] (in days/hours/mins/seconds).
'`@max`'	Maximum age (in days/hours/mins/seconds).
'`@mean`'	Arithmetic mean of ages (in days/hours/mins/seconds).
'`est+`'	Add low-high estimates.

You can also define custom summary types by setting `org-columns-summary-types`.

The '`est+`' summary type requires further explanation. It is used for combining estimates, expressed as low-high ranges. For example, instead of estimating a particular task will take 5 days, you might estimate it as 5–6 days if you're fairly confident you know how much work is required, or 1–10 days if you do not really know what needs to be done. Both ranges average at 5.5 days, but the first represents a more predictable delivery.

When combining a set of such estimates, simply adding the lows and highs produces an unrealistically wide result. Instead, '`est+`' adds the statistical mean and variance of the subtasks, generating a final estimate from the sum. For example, suppose you had ten tasks,

each of which was estimated at 0.5 to 2 days of work. Straight addition produces an estimate of 5 to 20 days, representing what to expect if everything goes either extremely well or extremely poorly. In contrast, ‘`est+`’ estimates the full job more realistically, at 10–15 days.

Here is an example for a complete columns definition, along with allowed values[56].

```
:COLUMNS:  %25ITEM %9Approved(Approved?){X} %Owner %11Status \
                  %10Time_Estimate{:} %CLOCKSUM %CLOCKSUM_T
:Owner_ALL:    Tammy Mark Karl Lisa Don
:Status_ALL:   "In progress" "Not started yet" "Finished" ""
:Approved_ALL: "[ ]" "[X]"
```

The first column, ‘`%25ITEM`’, means the first 25 characters of the item itself, i.e., of the headline. You probably always should start the column definition with the ‘`ITEM`’ specifier. The other specifiers create columns ‘`Owner`’ with a list of names as allowed values, for ‘`Status`’ with four different possible values, and for a checkbox field ‘`Approved`’. When no width is given after the ‘%’ character, the column is exactly as wide as it needs to be in order to fully display all values. The ‘`Approved`’ column does have a modified title (‘`Approved?`’, with a question mark). Summaries are created for the ‘`Time_Estimate`’ column by adding time duration expressions like HH:MM, and for the ‘`Approved`’ column, by providing an ‘`[X]`’ status if all children have been checked. The ‘`CLOCKSUM`’ and ‘`CLOCKSUM_T`’ columns are special, they lists the sums of CLOCK intervals in the subtree, either for all clocks or just for today.

Next: Capturing column view, Previous: Defining columns, Up: Column View [Contents][Index]

7.5.2 Using column view

Turning column view on or off

`C-c C-x C-c` (`org-columns`) ¶
Turn on column view. If point is before the first headline in the file, column view is turned on for the entire file, using the ‘`#+COLUMNS`’ definition. If point is somewhere inside the outline, this command searches the hierarchy, up from point, for a ‘`COLUMNS`’ property that defines a format. When one is found, the column view table is established for the tree starting at the entry that contains the ‘`COLUMNS`’ property. If no such property is found, the format is taken from the ‘`#+COLUMNS`’ line or from the variable `org-columns-default-format`, and column view is established for the current entry and its subtree.

`r` or `g` on a columns view line (`org-columns-redo`) ¶
Recreate the column view, to include recent changes made in the buffer.

`C-c C-c` or `q` on a columns view line (`org-columns-quit`) ¶
Exit column view.

Editing values

`LEFT`, `RIGHT`, `UP`, `DOWN`
Move through the column view from field to field.

`1..9,0` ¶
Directly select the Nth allowed value, `0` selects the 10th value.

n or **S-RIGHT** (**org-columns-next-allowed-value**)¶
p or **S-LEFT** (**org-columns-previous-allowed-value**)
Switch to the next/previous allowed value of the field. For this, you have to have specified allowed values for a property.

e (**org-columns-edit-value**)¶
Edit the property at point. For the special properties, this invokes the same interface that you normally use to change that property. For example, the tag completion or fast selection interface pops up when editing a ‘TAGS’ property.

C-c C-c (**org-columns-toggle-or-columns-quit**)¶
When there is a checkbox at point, toggle it. Else exit column view.

v (**org-columns-show-value**)¶
View the full value of this property. This is useful if the width of the column is smaller than that of the value.

a (**org-columns-edit-allowed**)¶
Edit the list of allowed values for this property. If the list is found in the hierarchy, the modified values is stored there. If no list is found, the new value is stored in the first entry that is part of the current column view.

Modifying column view on-the-fly

< (**org-columns-narrow**)¶
> (**org-columns-widen**)
Make the column narrower/wider by one character.

S-M-RIGHT (**org-columns-new**)¶
Insert a new column, to the left of the current column.

S-M-LEFT (**org-columns-delete**)¶
Delete the current column.

M-LEFT (**org-columns-move-left**)¶
Move the current column left.

M-RIGHT (**org-columns-move-right**)¶
Move the current column right.

M-UP (**org-columns-move-row-up**)¶
Move the current row up.

M-DOWN (**org-columns-move-row-down**)¶
Move the current row down.

7.5.3 Capturing column view

Since column view is just an overlay over a buffer, it cannot be exported or printed directly. If you want to capture a column view, use a ‘`columnview`’ dynamic block (see Dynamic Blocks). The frame of this block looks like this:

```
* The column view
#+BEGIN: columnview :hlines 1 :id "label"

#+END:
```

This dynamic block has the following parameters:

‘`:id`’
This is the most important parameter. Column view is a feature that is often localized to a certain (sub)tree, and the capture block might be at a different location in the file. To identify the tree whose view to capture, you can use four values:

‘`local`’
Use the tree in which the capture block is located.

‘`global`’
Make a global view, including all headings in the file.

‘`file:FILENAME`’
Run column view at the top of the *FILENAME* file.

‘`LABEL`’ ¶
Call column view in the tree that has an ‘`ID`’ property with the value *LABEL*. You can use `M-x org-id-copy` to create a globally unique ID for the current entry and copy it to the kill-ring.

‘`:match`’
When set to a string, use this as a tags/property match filter to select only a subset of the headlines in the scope set by the `:id` parameter.

‘`:hlines`’
When `t`, insert an hline after every line. When a number N, insert an hline before each headline with level `<= N`.

‘`:vlines`’
When non-`nil`, force column groups to get vertical lines.

‘`:maxlevel`’
When set to a number, do not capture entries below this level.

‘`:skip-empty-rows`’
When non-`nil`, skip rows where the only non-empty specifier of the column view is ‘`ITEM`’.

‘`:exclude-tags`’
List of tags to exclude from column view table: entries with these tags will be excluded from the column view.

‘`:indent`’
When non-`nil`, indent each ‘`ITEM`’ field according to its level.

‘`:link`’
When non-`nil`, link the ‘`ITEM`’ headlines in the table to their origins.

‘:format’
Specify a column attribute (see Column attributes) for the dynamic block.

‘:formatter’ ¶
A function to format column view data and insert it into the buffer. See the option `org-columns-dblock-formatter`.

The following commands insert or update the dynamic block:

`org-columns-insert-dblock` ¶
Insert a dynamic block capturing a column view. Prompt for the scope or ID of the view.

This command can be invoked by calling `org-dynamic-block-insert-dblock` (`C-c C-x x`) and selecting “columnview” (see Dynamic Blocks).

`C-c C-c` **`C-c C-x C-u`** (**`org-dblock-update`**) ¶
Update dynamic block at point. Point needs to be on the ‘`#+BEGIN`’ line of the dynamic block.

`C-u C-c C-x C-u` (**`org-update-all-dblocks`**) ¶
Update all dynamic blocks (see Dynamic Blocks). This is useful if you have several clock table blocks, column-capturing blocks or other dynamic blocks in a buffer.

You can add formulas to the column view table and you may add plotting instructions in front of the table—these survive an update of the block. If there is a ‘`TBLFM`’ keyword after the table, the table is recalculated automatically after an update.

An alternative way to capture and process property values into a table is provided by Eric Schulte’s ‘`org-collector.el`’, which is a package in ‘`org-contrib`’[57]. It provides a general API to collect properties from entries in a certain scope, and arbitrary Lisp expressions to process these values before inserting them into a table or a dynamic block.

---

8 Dates and Times

To assist project planning, TODO items can be labeled with a date and/or a time. The specially formatted string carrying the date and time information is called a *timestamp* in Org mode. This may be a little confusing because timestamp is often used as indicating when something was created or last changed. However, in Org mode this term is used in a much wider sense.

Timestamps can be used to plan appointments, schedule tasks, set deadlines, track time, and more. The following sections describe the timestamp format and tooling that Org mode provides for common use cases dealing with time and time intervals.

8.1 Timestamps

A timestamp is a specification of a date—possibly with a time or time range—in a special format, either ‘`<2003-09-16 Tue>`’ or ‘`<2003-09-16 Tue 09:39>`’ or ‘`<2003-09-16 Tue 12:00-12:30>`’[58]. A timestamp can appear anywhere in the headline or body of an Org tree entry. Its presence causes entries to be shown on specific dates in the agenda (see Weekly/daily agenda). We distinguish:

Plain timestamp; Event; Appointment ¶

A simple timestamp just assigns a date/time to an item. This is just like writing down an appointment or event in a paper agenda. In the agenda display, the headline of an entry associated with a plain timestamp is shown exactly on that date. There can be multiple timestamps in an item.

```
* Meet Peter at the movies
  <2006-11-01 Wed 19:15>
* Discussion on climate change
  <2006-11-02 Thu 10:00-12:00>
* My days off
  <2006-11-03 Fri>
  <2006-11-06 Mon>
```

Timestamp with repeater interval ¶

A timestamp may contain a *repeater interval*, indicating that it applies not only on the given date, but again and again after a certain interval of N hours (h), days (d), weeks (w), months (m), or years (y). The following shows up in the agenda every Wednesday:

```
* Pick up Sam at school
  <2007-05-16 Wed 12:30 +1w>
```

Diary-style expression entries ¶

For more complex date specifications, Org mode supports using the special expression diary entries implemented in the (emacs)Emacs Calendar package[59]. For example, with optional time:

```
* 22:00-23:00 The nerd meeting on every 2nd Thursday of the month
  <%%(diary-float t 4 2) 22:00-23:00>
```

Time range ¶

Time range is a timestamp having two time units connected by ‘-’

```
* Discussion on climate change
   <2006-11-02 Thu 10:00-12:00>
```

Time/Date range ¶

Two timestamps connected by ‘- -’ denote a range. In the agenda, the headline is shown on the first and last day of the range, and on any dates that are displayed and fall in the range. The first example specifies just the dates of the range while the second example specifies a time range for each date.

```
** Meeting in Amsterdam
   <2004-08-23 Mon>--<2004-08-26 Thu>
** This weeks committee meetings
   <2004-08-23 Mon 10:00-11:00>--<2004-08-26 Thu 10:00-11:00>
```

Inactive timestamp ¶

Just like a plain timestamp, but with square brackets instead of angular ones. These timestamps are inactive in the sense that they do *not* trigger an entry to show up in the agenda.

```
* Gillian comes late for the fifth time
  [2006-11-01 Wed]
```

8.2 Creating Timestamps

For Org mode to recognize timestamps, they need to be in the specific format. All commands listed below produce timestamps in the correct format.

C-c . (**org-timestamp**) ¶
Prompt for a date and insert a corresponding timestamp. When point is at an existing timestamp in the buffer, the command is used to modify this timestamp instead of inserting a new one. When this command is used twice in succession, a time range is inserted.

When called with a prefix argument, use the alternative format which contains date and time. The default time can be rounded to multiples of 5 minutes. See the option `org-timestamp-rounding-minutes`.

With two prefix arguments, insert an active timestamp with the current time without prompting.

C-c ! (**org-timestamp-inactive**) ¶
Like `C-c .`, but insert an inactive timestamp that does not cause an agenda entry.

C-c C-c ¶
Normalize timestamp, insert or fix day name if missing or wrong.

C-c < (**org-date-from-calendar**) ¶
Insert a timestamp corresponding to point date in the calendar.

C-c > (**org-goto-calendar**) ¶
Access the Emacs calendar for the current date. If there is a timestamp in the current line, go to the corresponding date instead.

C-c C-o (**org-open-at-point**) ¶
Access the agenda for the date given by the timestamp or -range at point (see Weekly/daily agenda).

S-LEFT (**org-timestamp-down-day**) ¶
S-RIGHT (**org-timestamp-up-day**)
Change date at point by one day. These key bindings conflict with shift-selection and related modes (see Packages that conflict with Org mode).

S-UP (**org-timestamp-up**) ¶
S-DOWN (**org-timestamp-down**)
On the beginning or enclosing bracket of a timestamp, change its type. Within a timestamp, change the item under point. Point can be on a year, month, day, hour or minute. When the timestamp contains a time range like '`15:30-16:30`', modifying the first time also shifts the second, shifting the time block with constant length. To change the length, modify the second time. Note that if point is in a headline and not at a timestamp, these same keys modify the priority of an item (see Priorities). The key bindings also conflict with shift-selection and related modes (see Packages that conflict with Org mode).

C-c C-y (**org-evaluate-time-range**) ¶
Evaluate a time range by computing the difference between start and end. With a prefix argument, insert result after the time range (in a table: into the following column).

8.2.1 The date/time prompt

When Org mode prompts for a date/time, the default is shown in default date/time format, and the prompt therefore seems to ask for a specific format. But it in fact accepts date/time information in a variety of formats. Generally, the information should start at the beginning of the string. Org mode finds whatever information is in there and derives anything you have not specified from the *default date and time*. The default is usually the current date and time, but when modifying an existing timestamp, or when entering the second stamp of a range, it is taken from the stamp in the buffer. When filling in information, Org mode assumes that most of the time you want to enter a date in the future: if you omit the month/year and the given day/month is *before* today, it assumes that you mean a future date[60]. If the date has been automatically shifted into the future, the time prompt shows this with '`(=>F)`'.

For example, let's assume that today is **June 13, 2006**. Here is how various inputs are interpreted, the items filled in by Org mode are in **bold**.

'`3-2-5`'	⇒ 2003-02-05
'`2/5/3`'	⇒ 2003-02-05
'`14`'	⇒ **2006**-**06**-14
'`12`'	⇒ **2006**-**07**-12
'`2/5`'	⇒ **2007**-02-05
'`Fri`'	⇒ nearest Friday (default date or later)
'`sep 15`'	⇒ **2006**-09-15
'`feb 15`'	⇒ **2007**-02-15
'`sep 12 9`'	⇒ 2009-09-12
'`12:45`'	⇒ **2006**-**06**-**13** 12:45
'`22 sept 0:34`'	⇒ **2006**-09-22 0:34
'`w4`'	⇒ ISO week for of the current year **2006**
'`2012 w4 fri`'	⇒ Friday of ISO week 4 in 2012
'`2012-w04-5`'	⇒ Same as above

Furthermore you can specify a relative date by giving, as the *first* thing in the input: a plus/minus sign, a number and a letter—'`h`', '`d`', '`w`', '`m`' or '`y`'—to indicate a change in hours, days, weeks, months, or years. With '`h`' the date is relative to the current time, with the other letters and a single plus or minus, the date is relative to today at 00:00. With a double plus or minus, it is relative to the default date. If instead of a single letter, you use the abbreviation of day name, the date is the Nth such day, e.g.:

'`+0`'	⇒ today

'.'	⇒ today
'+2h'	⇒ two hours from now
'+4d'	⇒ four days from today
'+4'	⇒ same as +4d
'+2w'	⇒ two weeks from today
'++5'	⇒ five days from default date
'+2tue'	⇒ second Tuesday from now

The function understands English month and weekday abbreviations. If you want to use un-abbreviated names and/or other languages, configure the variables `parse-time-months` and `parse-time-weekdays`.

Not all dates can be represented in a given Emacs implementation. By default Org mode forces dates into the compatibility range 1970–2037 which works on all Emacs implementations. If you want to use dates outside of this range, read the docstring of the variable `org-read-date-force-compatible-dates`.

You can specify a time range by giving start and end times or by giving a start time and a duration (in HH:MM format). Use one or two dash(es) as the separator in the former case and use '+' as the separator in the latter case, e.g.:

'11am-1:15pm'	⇒ 11:00-13:15
'11h-13h15'	⇒ same as above
'11am--1:15pm'	⇒ same as above
'11am+2:15'	⇒ same as above

Parallel to the minibuffer prompt, a calendar is popped up[61]. When you exit the date prompt, either by clicking on a date in the calendar, or by pressing `RET`, the date selected in the calendar is combined with the information entered at the prompt. You can control the calendar fully from the minibuffer:

`RET`	Choose date at point in calendar.
`mouse-1`	Select date by clicking on it.
`S-RIGHT`	One day forward.
`S-LEFT`	One day backward.
`S-DOWN`	One week forward.
`S-UP`	One week backward.
`M-S-RIGHT`	One month forward.

`M-S-LEFT`	One month backward.
`>`	Scroll calendar forward by one month.
`<`	Scroll calendar backward by one month.
`M-v`	Scroll calendar forward by 3 months.
`C-v`	Scroll calendar backward by 3 months.
`C-.`	Select today's date[62]

The actions of the date/time prompt may seem complex, but I assure you they will grow on you, and you will start getting annoyed by pretty much any other way of entering a date/time out there. To help you understand what is going on, the current interpretation of your input is displayed live in the minibuffer[63].

8.2.2 Custom time format

Org mode uses the standard ISO notation for dates and times as it is defined in ISO 8601. If you cannot get used to this and require another representation of date and time to keep you happy, you can get it by customizing the variables `org-display-custom-times` and `org-timestamp-custom-formats`.

`C-c C-x C-t` (**`org-toggle-timestamp-overlays`**) ¶
Toggle the display of custom formats for dates and times.

Org mode needs the default format for scanning, so the custom date/time format does not *replace* the default format. Instead, it is put *over* the default format using text properties. This has the following consequences:

- You cannot place point onto a timestamp anymore, only before or after.
- The `S-UP` and `S-DOWN` keys can no longer be used to adjust each component of a timestamp. If point is at the beginning of the stamp, `S-UP` and `S-DOWN` change the stamp by one day, just like `S-LEFT` `S-RIGHT`. At the end of the stamp, change the time by one minute.
- If the timestamp contains a range of clock times or a repeater, these are not overlaid, but remain in the buffer as they were.
- When you delete a timestamp character-by-character, it only disappears from the buffer after *all* (invisible) characters belonging to the ISO timestamp have been removed.
- If the custom timestamp format is longer than the default and you are using dates in tables, table alignment will be messed up. If the custom format is shorter, things do work as expected.

8.3 Deadlines and Scheduling

A timestamp may be preceded by special keywords to facilitate planning. Both the timestamp and the keyword have to be positioned immediately after the task they refer to.

'`DEADLINE`' ¶
Meaning: the task—most likely a TODO item, though not necessarily—is supposed to be finished on that date.

On the deadline date, the task is listed in the agenda. In addition, the agenda for *today* carries a warning about the approaching or missed deadline, starting `org-deadline-warning-days` before the due date, and continuing until the entry is marked as done. An example:

```
*** TODO write article about the Earth for the Guide
    DEADLINE: <2004-02-29 Sun>
    The editor in charge is [[bbdb:Ford Prefect]]
```

You can specify a different lead time for warnings for a specific deadlines using the following syntax. Here is an example with a warning period of 5 days '`DEADLINE: <2004-02-29 Sun -5d>`'. This warning is deactivated if the task gets scheduled and you set `org-agenda-skip-deadline-prewarning-if-scheduled` to `t`.

'SCHEDULED' ¶
Meaning: you are planning to start working on that task on the given date.

The headline is listed under the given date[64]. In addition, a reminder that the scheduled date has passed is present in the compilation for *today*, until the entry is marked as done, i.e., the task is automatically forwarded until completed.

```
*** TODO Call Trillian for a date on New Years Eve.
    SCHEDULED: <2004-12-25 Sat>
```

If you want to *delay* the display of this task in the agenda, use '`SCHEDULED: <2004-12-25 Sat -2d>`': the task is still scheduled on the 25th but will appear two days later. In case the task contains a repeater, the delay is considered to affect all occurrences; if you want the delay to only affect the first scheduled occurrence of the task, use '`--2d`' instead. See `org-scheduled-delay-days` and `org-agenda-skip-scheduled-delay-if-deadline` for details on how to control this globally or per agenda.

> **Important:** Scheduling an item in Org mode should *not* be understood in the same way that we understand *scheduling a meeting*. Setting a date for a meeting is just a simple appointment, you should mark this entry with a simple plain timestamp, to get this item shown on the date where it applies. This is a frequent misunderstanding by Org users. In Org mode, *scheduling* means setting a date when you want to start working on an action item.

You may use timestamps with repeaters in scheduling and deadline entries. Org mode issues early and late warnings based on the assumption that the timestamp represents the *nearest instance* of the repeater. However, the use of diary expression entries like

```
<%%(diary-float t 42)>
```

in scheduling and deadline timestamps is limited. Org mode does not know enough about the internals of each function to issue early and late warnings. However, it shows the item on each day where the expression entry matches.

8.3.1 Inserting deadlines or schedules

The following commands allow you to quickly insert a deadline or to schedule an item:[65]

`C-c C-d` (**`org-deadline`**) ¶

Insert ‘DEADLINE’ keyword along with a stamp. The insertion happens in the line directly following the headline. Remove any ‘CLOSED’ timestamp . When called with a prefix argument, also remove any existing deadline from the entry. Depending on the variable `org-log-redeadline`, take a note when changing an existing deadline[66].

`C-c C-s` (`org-schedule`) ¶
Insert ‘SCHEDULED’ keyword along with a stamp. The insertion happens in the line directly following the headline. Remove any ‘CLOSED’ timestamp. When called with a prefix argument, also remove the scheduling date from the entry. Depending on the variable `org-log-reschedule`, take a note when changing an existing scheduling time[67].

`C-c / d` (`org-check-deadlines`) ¶
Create a sparse tree with all deadlines that are either past-due, or which will become due within `org-deadline-warning-days`. With `C-u` prefix, show all deadlines in the file. With a numeric prefix, check that many days. For example, `C-1 C-c / d` shows all deadlines due tomorrow.

`C-c / b` (`org-check-before-date`) ¶
Sparse tree for deadlines and scheduled items before a given date.

`C-c / a` (`org-check-after-date`) ¶
Sparse tree for deadlines and scheduled items after a given date.

Note that `org-schedule` and `org-deadline` supports setting the date by indicating a relative time e.g., ‘+1d’ sets the date to the next day after today, and ‘--1w’ sets the date to the previous week before any current timestamp.

8.3.2 Repeated tasks

Some tasks need to be repeated again and again. Org mode helps to organize such tasks using a so-called repeater in a ‘DEADLINE’, ‘SCHEDULED’, or plain timestamps[68]. In the following example:

```
** TODO Pay the rent
   DEADLINE: <2005-10-01 Sat +1m>
```

the ‘+1m’ is a repeater; the intended interpretation is that the task has a deadline on ‘<2005-10-01>’ and repeats itself every (one) month starting from that time. You can use yearly, monthly, weekly, daily and hourly repeat cookies by using the ‘y’, ‘m’, ‘w’, ‘d’ and ‘h’ letters. If you need both a repeater and a special warning period in a deadline entry, the repeater should come first and the warning period last

```
DEADLINE: <2005-10-01 Sat +1m -3d>
```

Deadlines and scheduled items produce entries in the agenda when they are over-due, so it is important to be able to mark such an entry as done once you have done so. When you mark a ‘DEADLINE’ or a ‘SCHEDULED’ with the TODO keyword ‘DONE’, it no longer produces entries in the agenda. The problem with this is, however, is that then also the *next* instance of the repeated entry will not be active. Org mode deals with this in the following way: when you try to mark such an entry as done, using `C-c C-t`, it shifts the base date of the repeating timestamp by the repeater interval, and immediately sets the entry state back to TODO[69]. In the example above, setting the state to ‘DONE’ would actually switch the date like this:

```
** TODO Pay the rent
   DEADLINE: <2005-11-01 Tue +1m>
```

When task contains multiple timestamps with repeater interval, all these timestamps are shifted.

To mark a task with a repeater as DONE, use `C-- 1 C-c C-t`, i.e., `org-todo` with a numeric prefix argument of '`-1`'.

A timestamp[70] is added under the deadline, to keep a record that you actually acted on the previous instance of this deadline.

As a consequence of shifting the base date, this entry is no longer visible in the agenda when checking past dates, but all future instances will be visible.

With the '`+1m`' cookie, the date shift is always exactly one month. So if you have not paid the rent for three months, marking this entry DONE still keeps it as an overdue deadline. Depending on the task, this may not be the best way to handle it. For example, if you forgot to call your father for 3 weeks, it does not make sense to call him 3 times in a single day to make up for it. Finally, there are tasks, like changing batteries, which should always repeat a certain time *after* the last time you did it. For these tasks, Org mode has special repeaters '++' and '`.+`'. For example:

```
** TODO Call Father
   DEADLINE: <2008-02-10 Sun ++1w>
   Marking this DONE shifts the date by at least one week, but also
   by as many weeks as it takes to get this date into the future.
   However, it stays on a Sunday, even if you called and marked it
   done on Saturday.

** TODO Empty kitchen trash
   DEADLINE: <2008-02-08 Fri 20:00 ++1d>
   Marking this DONE shifts the date by at least one day, and also
   by as many days as it takes to get the timestamp into the future.
   Since there is a time in the timestamp, the next deadline in the
   future will be on today's date if you complete the task before
   20:00.

** TODO Check the batteries in the smoke detectors
   DEADLINE: <2005-11-01 Tue .+1m>
   Marking this DONE shifts the date to one month after today.

** TODO Wash my hands
   DEADLINE: <2019-04-05 08:00 Fri .+1h>
   Marking this DONE shifts the date to exactly one hour from now.
```

You may have both scheduling and deadline information for a specific task. If the repeater is set for the scheduling information only, you probably want the repeater to be ignored after the deadline. If so, set the variable `org-agenda-skip-scheduled-repeats-after-deadline` to `t`. However, any scheduling information without a repeater is no longer relevant once the task is done, and thus, removed upon repeating the task. If you want both scheduling and deadline information to repeat after the same interval, set the same repeater for both timestamps.

An alternative to using a repeater is to create a number of copies of a task subtree, with dates shifted in each copy. The command `C-c C-x c` was created for this purpose; it is described in Structure Editing.

8.4 Clocking Work Time

Org mode allows you to clock the time you spend on specific tasks in a project. When you start working on an item, you can start the clock. When you stop working on that task, or when you mark the task done, the clock is stopped and the corresponding time interval is recorded. It also computes the total time spent on each subtree[71] of a project. And it remembers a history or tasks recently clocked, so that you can jump quickly between a number of tasks absorbing your time.

To save the clock history across Emacs sessions, use:

```
(setq org-clock-persist 'history)
(org-clock-persistence-insinuate)
```

When you clock into a new task after resuming Emacs, the incomplete clock[72] is retrieved (see Resolving idle time (1)) and you are prompted about what to do with it.

8.4.1 Clocking commands

`C-c C-x C-i` (**`org-clock-in`**) ¶
Start the clock on the current item (clock-in). This inserts the 'CLOCK' keyword together with a timestamp. If this is not the first clocking of this item, the multiple 'CLOCK' lines are wrapped into a 'LOGBOOK' drawer (see also the variable `org-clock-into-drawer`). You can also overrule the setting of this variable for a subtree by setting a 'CLOCK_INTO_DRAWER' or 'LOG_INTO_DRAWER' property. When called with a `C-u` prefix argument, select the task from a list of recently clocked tasks. With two `C-u C-u` prefixes, clock into the task at point and mark it as the default task; the default task is always be available with letter `d` when selecting a clocking task. With three `C-u C-u C-u` prefixes, force continuous clocking by starting the clock when the last clock stopped.

While the clock is running, Org shows the current clocking time in the mode line, along with the title of the task. The clock time shown is all time ever clocked for this task and its children. If the task has an effort estimate (see Effort Estimates), the mode line displays the current clocking time against it[73]. If the task is a repeating one (see Repeated tasks), show only the time since the last reset of the task[74]. You can exercise more control over show time with the 'CLOCK_MODELINE_TOTAL' property. It may have the values '`current`' to show only the current clocking instance, '`today`' to show all time clocked on this tasks today—see also the variable `org-extend-today-until`, `all` to include all time, or `auto` which is the default[75]. Clicking with `mouse-1` onto the mode line entry pops up a menu with clocking options.

`C-c C-x C-o` (**`org-clock-out`**) ¶
Stop the clock (clock-out). This inserts another timestamp at the same location where the clock was last started. It also directly computes the resulting time in inserts it after the time range as '=>HH:MM'. See the variable `org-log-note-clock-out` for the possibility to record an additional note together with the clock-out timestamp[76].

`C-c C-x C-x` (`org-clock-in-last`) ¶
Re-clock the last clocked task. With one `C-u` prefix argument, select the task from the clock history. With two `C-u` prefixes, force continuous clocking by starting the clock when the last clock stopped.

`C-c C-x C-e` (`org-clock-modify-effort-estimate`) ¶
Update the effort estimate for the current clock task.

`C-c C-c` or `C-c C-y` (`org-evaluate-time-range`) ¶
Recompute the time interval after changing one of the timestamps. This is only necessary if you edit the timestamps directly. If you change them with `S-<cursor>` keys, the update is automatic.

`C-S-UP` (`org-clock-timestamps-up`) ¶
`C-S-DOWN` (`org-clock-timestamps-down`)
On CLOCK log lines, increase/decrease both timestamps so that the clock duration keeps the same value.

`S-M-UP` (`org-timestamp-up`) ¶
`S-M-DOWN` (`org-timestamp-down`)
On ‘CLOCK’ log lines, increase/decrease the timestamp at point and the one of the previous, or the next, clock timestamp by the same duration. For example, if you hit `S-M-UP` to increase a clocked-out timestamp by five minutes, then the clocked-in timestamp of the next clock is increased by five minutes.

Only ‘CLOCK’ logs created during current Emacs session are considered when adjusting next/previous timestamp.

`C-c C-t` (`org-todo`) ¶
Changing the TODO state of an item to DONE automatically stops the clock if it is running in this same item.

`C-c C-x C-q` (`org-clock-cancel`) ¶
Cancel the current clock. This is useful if a clock was started by mistake, or if you ended up working on something else.

`C-c C-x C-j` (`org-clock-goto`) ¶
Jump to the headline of the currently clocked in task. With a `C-u` prefix argument, select the target task from a list of recently clocked tasks.

`C-c C-x C-d` (`org-clock-display`) ¶
Display time summaries for each subtree in the current buffer. This puts overlays at the end of each headline, showing the total time recorded under that heading, including the time of any subheadings. You can use visibility cycling to study the tree, but the overlays disappear when you change the buffer (see variable `org-remove-highlights-with-change`) or press `C-c C-c`.

The `l` key may be used in the agenda (see Weekly/daily agenda) to show which tasks have been worked on or closed during a day.

Important: note that both `org-clock-out` and `org-clock-in-last` can have a global keybinding and do not modify the window disposition.

8.4.2 The clock table

Org mode can produce quite complex reports based on the time clocking information. Such a report is called a *clock table*, because it is formatted as one or several Org tables.

`org-clock-report` ¶
Insert or update a clock table. When called with a prefix argument, jump to the first clock table in the current document and update it. The clock table includes archived trees.

This command can be invoked by calling `org-dynamic-block-insert-dblock` (`C-c C-x x`) and selecting "clocktable" (see Dynamic Blocks).

`C-c C-c` or **`C-c C-x C-u`** (**`org-dblock-update`**) ¶
Update dynamic block at point. Point needs to be in the 'BEGIN' line of the dynamic block.

`C-u C-c C-x C-u` ¶
Update all dynamic blocks (see Dynamic Blocks). This is useful if you have several clock table blocks in a buffer.

`S-LEFT` ¶
`S-RIGHT` (**`org-clocktable-try-shift`**)
Shift the current ':block' interval and update the table. Point needs to be in the '#+BEGIN: clocktable' line for this command. If ':block' is 'today', it is shifted to 'today-1', etc.

Here is an example of the frame for a clock table as it is inserted into the buffer by `org-clock-report`:

```
#+BEGIN: clocktable :maxlevel 2 :emphasize nil :scope file
#+END:
```

The '#+BEGIN' line contains options to define the scope, structure, and formatting of the report. Defaults for all these options can be configured in the variable `org-clocktable-defaults`.

First there are options that determine which clock entries are to be selected:

':maxlevel'
Maximum level depth to which times are listed in the table. Clocks at deeper levels are summed into the upper level.

':scope'
The scope to consider. This can be any of the following:

'nil'	the current buffer or narrowed region
'file'	the full current buffer
'subtree'	the subtree where the clocktable is located
'treeN'	the surrounding level N tree, for example 'tree3'

‘`tree`’	the surrounding level 1 tree
‘`agenda`’	all agenda files
‘`("file" ...)`’	scan these files
‘`FUNCTION`’	scan files returned by calling *FUNCTION* with no argument
‘`file-with-archives`’	current file and its archives
‘`agenda-with-archives`’	all agenda files, including archives

‘`:block`’
The time block to consider. This block is specified either absolutely, or relative to the current time and may be any of these formats:

‘`2007-12-31`’	New year eve 2007
‘`2007-12`’	December 2007
‘`2007-W50`’	ISO-week 50 in 2007
‘`2007-Q2`’	2nd quarter in 2007
‘`2007`’	the year 2007
‘`today`’, ‘`yesterday`’, ‘`today-N`’	a relative day
‘`thisweek`’, ‘`lastweek`’, ‘`thisweek-N`’	a relative week
‘`thismonth`’, ‘`lastmonth`’, ‘`thismonth-N`’	a relative month
‘`thisyear`’, ‘`lastyear`’, ‘`thisyear-N`’	a relative year
‘`untilnow`’[77]	all clocked time ever

When this option is not set, Org falls back to the value in `org-clock-display-default-range`, which defaults to the current year.

Use `S-LEFT` or `S-RIGHT` to shift the time interval.

‘`:tstart`’
A time string specifying when to start considering times. Relative times like ‘`"<-2w>"`’ can also be used. See Matching tags and properties for relative time syntax.

‘`:tend`’
A time string specifying when to stop considering times. Relative times like ‘`"<now>"`’ can also be used. See Matching tags and properties for relative time syntax.

‘`:wstart`’
The starting day of the week. The default is 1 for Monday.

‘`:mstart`’
The starting day of the month. The default is 1 for the first.

‘:step’
Set to ‘day’, ‘week’, ‘semimonth’, ‘month’, ‘quarter’, or ‘year’ to split the table into chunks. To use this, either ‘:block’, or ‘:tstart’ and ‘:tend’ are required.

‘:stepskip0’
When non-nil, do not show steps that have zero time.

‘:fileskip0’
When non-nil, do not show table sections from files which did not contribute.

‘:match’
A tags match to select entries that should contribute. See Matching tags and properties for the match syntax.

Then there are options that determine the formatting of the table. There options are interpreted by the function org-clocktable-write-default, but you can specify your own function using the ‘:formatter’ parameter.

‘:emphasize’
When non-nil, emphasize level one and level two items.

‘:lang’
Language[78] to use for descriptive cells like “Task”.

‘:link’
Link the item headlines in the table to their origins.

‘:narrow’
An integer to limit the width of the headline column in the Org table. If you write it like ‘50!’, then the headline is also shortened in export.

‘:indent’
Indent each headline field according to its level.

‘:filetitle’
Show title in the file column if the file has a ‘#+title’.

‘:hidefiles’
Hide the file column when multiple files are used to produce the table.

‘:tcolumns’
Number of columns to be used for times. If this is smaller than ‘:maxlevel’, lower levels are lumped into one column.

‘:level’
Should a level number column be included?

‘:sort’
A cons cell containing the column to sort and a sorting type. E.g., ‘:sort (1 . ?a)’ sorts the first column alphabetically.

‘:compact’
Abbreviation for ‘`:level nil :indent t :narrow 40! :tcolumns 1`’. All are overwritten except if there is an explicit ‘`:narrow`’.

‘:timestamp’
A timestamp for the entry, when available. Look for ‘`SCHEDULED`’, ‘`DEADLINE`’, ‘`TIMESTAMP`’ and ‘`TIMESTAMP_IA`’ special properties (see Special Properties), in this order.

‘:tags’
When this flag is non-`nil`, show the headline’s tags.

‘:properties’
List of properties shown in the table. Each property gets its own column.

‘:inherit-props’
When this flag is non-`nil`, the values for ‘`:properties`’ are inherited.

‘:formula’
Content of a ‘`TBLFM`’ keyword to be added and evaluated. As a special case, ‘`:formula %`’ adds a column with % time. If you do not specify a formula here, any existing formula below the clock table survives updates and is evaluated.

‘:formatter’
A function to format clock data and insert it into the buffer.

To get a clock summary of the current level 1 tree, for the current day, you could write:

```
#+BEGIN: clocktable :maxlevel 2 :block today :scope tree1 :link t
#+END:
```

To use a specific time range you could write[79]

```
#+BEGIN: clocktable :tstart "<2006-08-10 Thu 10:00>"
                    :tend "<2006-08-10 Thu 12:00>"
#+END:
```

A range starting a week ago and ending right now could be written as

```
#+BEGIN: clocktable :tstart "<-1w>" :tend "<now>"
#+END:
```

A summary of the current subtree with % times would be

```
#+BEGIN: clocktable :scope subtree :link t :formula %
#+END:
```

A horizontally compact representation of everything clocked during last week would be

```
#+BEGIN: clocktable :scope agenda :block lastweek :compact t
#+END:
```

Previous: The clock table, Up: Clocking Work Time [Contents][Index]

8.4.3 Resolving idle time and continuous clocking

Resolving idle time

If you clock in on a work item, and then walk away from your computer—perhaps to take a phone call—you often need to "resolve" the time you were away by either subtracting it from the current clock, or applying it to another one.

By customizing the variable `org-clock-idle-time` to some integer, such as 10 or 15, Emacs can alert you when you get back to your computer after being idle for that many minutes[80], and ask what you want to do with the idle time. There will be a question waiting for you when you get back, indicating how much idle time has passed constantly updated with the current amount, as well as a set of choices to correct the discrepancy:

k ¶
To keep some or all of the minutes and stay clocked in, press k. Org asks how many of the minutes to keep. Press RET to keep them all, effectively changing nothing, or enter a number to keep that many minutes.

K ¶
If you use the shift key and press K, it keeps however many minutes you request and then immediately clock out of that task. If you keep all of the minutes, this is the same as just clocking out of the current task.

s ¶
To keep none of the minutes, use s to subtract all the away time from the clock, and then check back in from the moment you returned.

S ¶
To keep none of the minutes and just clock out at the start of the away time, use the shift key and press S. Remember that using shift always leave you clocked out, no matter which option you choose.

C ¶
To cancel the clock altogether, use C. Note that if instead of canceling you subtract the away time, and the resulting clock amount is less than a minute, the clock is still canceled rather than cluttering up the log with an empty entry.

What if you subtracted those away minutes from the current clock, and now want to apply them to a new clock? Simply clock in to any task immediately after the subtraction. Org will notice that you have subtracted time "on the books", so to speak, and will ask if you want to apply those minutes to the next task you clock in on.

There is one other instance when this clock resolution magic occurs. Say you were clocked in and hacking away, and suddenly your cat chased a mouse who scared a hamster that crashed into your UPS's power button! You suddenly lose all your buffers, but thanks to auto-save you still have your recent Org mode changes, including your last clock in.

If you restart Emacs and clock into any task, Org will notice that you have a dangling clock which was never clocked out from your last session. Using that clock's starting time as the beginning of the unaccounted-for period, Org will ask how you want to resolve that time. The

logic and behavior is identical to dealing with away time due to idleness; it is just happening due to a recovery event rather than a set amount of idle time.

You can also check all the files visited by your Org agenda for dangling clocks at any time using `M-x org-resolve-clocks RET` (or `C-c C-x C-z`).

Continuous clocking

You may want to start clocking from the time when you clocked out the previous task. To enable this systematically, set `org-clock-continuously` to non-`nil`. Each time you clock in, Org retrieves the clock-out time of the last clocked entry for this session, and start the new clock from there.

If you only want this from time to time, use three universal prefix arguments with `org-clock-in` and two `C-u C-u` with `org-clock-in-last`.

Clocking out automatically after some idle time

When you often forget to clock out before being idle and you don't want to manually set the clocking time to take into account, you can set `org-clock-auto-clockout-timer` to a number of seconds and add '`(org-clock-auto-clockout-insinuate)`' to your '`.emacs`' file.

When the clock is running and Emacs is idle for more than this number of seconds, the clock will be clocked out automatically.

Use '`M-x org-clock-toggle-auto-clockout RET`' to temporarily turn this on or off.

8.5 Effort Estimates

If you want to plan your work in a very detailed way, or if you need to produce offers with quotations of the estimated work effort, you may want to assign effort estimates to entries. If you are also clocking your work, you may later want to compare the planned effort with the actual working time, a great way to improve planning estimates.

Effort estimates are stored in a special property '`EFFORT`'. Multiple formats are supported, such as '`3:12`', '`1:23:45`', or '`1d3h5min`'; see the file '`org-duration.el`' for more detailed information about the format.

You can set the effort for an entry with the following commands:

`C-c C-x e` (**`org-set-effort`**) ¶
Set the effort estimate for the current entry. With a prefix argument, set it to the next allowed value—see below. This command is also accessible from the agenda with the `e` key.

`C-c C-x C-e` (**`org-clock-modify-effort-estimate`**) ¶
Modify the effort estimate of the item currently being clocked.

Clearly the best way to work with effort estimates is through column view (see Column View). You should start by setting up discrete values for effort estimates, and a '`COLUMNS`' format that displays these values together with clock sums—if you want to clock your time. For a specific buffer you can use:

```
#+PROPERTY: Effort_ALL 0 0:10 0:30 1:00 2:00 3:00 4:00 5:00 6:00 7:00
#+COLUMNS: %40ITEM(Task) %17Effort(Estimated Effort){:} %CLOCKSUM
```

or, even better, you can set up these values globally by customizing the variables `org-global-properties` and `org-columns-default-format`. In particular if you want to use this setup also in the agenda, a global setup may be advised.

The way to assign estimates to individual items is then to switch to column mode, and to use `S-RIGHT` and `S-LEFT` to change the value. The values you enter are immediately summed up in the hierarchy. In the column next to it, any clocked time is displayed.

If you switch to column view in the daily/weekly agenda, the effort column summarizes the estimated work effort for each day[81], and you can use this to find space in your schedule. To get an overview of the entire part of the day that is committed, you can set the option `org-agenda-columns-add-appointments-to-effort-sum`. The appointments on a day that take place over a specified time interval are then also added to the load estimate of the day.

Effort estimates can be used in secondary agenda filtering that is triggered with the `/` key in the agenda (see Commands in the Agenda Buffer). If you have these estimates defined consistently, two or three key presses narrow down the list to stuff that fits into an available time slot.

8.6 Taking Notes with a Relative Timer

Org provides two types of timers. There is a relative timer that counts up, which can be useful when taking notes during, for example, a meeting or a video viewing. There is also a countdown timer.

The relative and countdown are started with separate commands.

`C-c C-x 0` (**`org-timer-start`**) ¶
Start or reset the relative timer. By default, the timer is set to 0. When called with a `C-u` prefix, prompt the user for a starting offset. The prompt will default to a timer string at point (if any), providing a convenient way to restart taking notes after a break in the process. When called with a double prefix argument `C-u C-u`, change all timer strings in the active region by a certain amount. This can be used to fix timer strings if the timer was not started at exactly the right moment.

`C-c C-x ;` (**`org-timer-set-timer`**) ¶
Start a countdown timer. The user is prompted for a duration. `org-timer-default-timer` sets the default countdown value. Giving a numeric prefix argument overrides this default value. This command is available as `;` in agenda buffers.

Once started, relative and countdown timers are controlled with the same commands.

`C-c C-x .` (**`org-timer`**) ¶
Insert a relative time into the buffer. The first time you use this, the timer starts. Using a prefix argument restarts it.

`C-c C-x -` (**`org-timer-item`**) ¶

Insert a description list item with the current relative time. With a prefix argument, first reset the timer to 0.

`M-RET` (`org-insert-heading`) ¶
Once the timer list is started, you can also use `M-RET` to insert new timer items.

`C-c C-x ,` (`org-timer-pause-or-continue`) ¶
Pause the timer, or continue it if it is already paused.

`C-c C-x _` (`org-timer-stop`) ¶
Stop the timer. After this, you can only start a new timer, not continue the old one. This command also removes the timer from the mode line.

9 Refiling and Archiving

Once information is in the system, it may need to be moved around. Org provides Refile, Copy and Archive commands for this. Refile and Copy helps with moving and copying outlines. Archiving helps to keep the system compact and fast.

- Refile and Copy
- Archiving

9.1 Refile and Copy

When reviewing the captured data, you may want to refile or to copy some of the entries into a different list, for example into a project. Cutting, finding the right location, and then pasting the note is cumbersome. To simplify this process, you can use the following special command:

`C-c C-w` (`org-refile`) ¶
Refile the entry or region at point. This command offers possible locations for refiling the entry and lets you select one with completion. The item (or all items in the region) is filed below the target heading as a subitem. Depending on `org-reverse-note-order`, it is either the first or last subitem.

By default, all level 1 headlines in the current buffer are considered to be targets, but you can have more complex definitions across a number of files. See the variable `org-refile-targets` for details. If you would like to select a location via a file-path-like completion along the outline path, see the variables `org-refile-use-outline-path` and `org-outline-path-complete-in-steps`. If you would like to be able to create new nodes as new parents for refiling on the fly, check the variable `org-refile-allow-creating-parent-nodes`. When the variable `org-log-refile`[82] is set, a timestamp or a note is recorded whenever an entry is refiled.

`C-u C-c C-w` ¶
Use the refile interface to jump to a heading.

`C-u C-u C-c C-w` (`org-refile-goto-last-stored`) ¶
Jump to the location where `org-refile` last moved a tree to.

`C-2 C-c C-w` ¶
Refile as the child of the item currently being clocked.

`C-3 C-c C-w` ¶
Refile and keep the entry in place. Also see `org-refile-keep` to make this the default behavior, and beware that this may result in duplicated ‘`ID`’ properties.

`C-0 C-c C-w` or `C-u C-u C-u C-c C-w` (`org-refile-cache-clear`) ¶
Clear the target cache. Caching of refile targets can be turned on by setting `org-refile-use-cache`. To make the command see new possible targets, you have to clear the cache with this command.

`C-c M-w` (`org-refile-copy`) ¶
Copying works like refiling, except that the original note is not deleted.

`C-c C-M-w` (`org-refile-reverse`) ¶
Works like refiling, except that it temporarily toggles how the value of `org-reverse-note-order` applies to the current buffer. So if `org-refile` would append the entry as the last entry under the target header, `org-refile-reverse` will prepend it as the first entry, and vice-versa.

9.2 Archiving

When a project represented by a (sub)tree is finished, you may want to move the tree out of the way and to stop it from contributing to the agenda. Archiving is important to keep your working files compact and global searches like the construction of agenda views fast.

`C-c C-x C-a` (`org-archive-subtree-default`) ¶
Archive the current entry using the command specified in the variable `org-archive-default-command`.

9.2.1 Moving a tree to an archive file

The most common archiving action is to move a project tree to another file, the archive file.

`C-c C-x C-s` or short `C-c $` (`org-archive-subtree`) ¶
Archive the subtree starting at point position to the location given by `org-archive-location`.

`C-u C-c C-x C-s` ¶
Check if any direct children of the current headline could be moved to the archive. To do this, check each subtree for open TODO entries. If none is found, the command offers to move it to the archive location. If point is *not* on a headline when this command is invoked, check level 1 trees.

`C-u C-u C-c C-x C-s` ¶
As above, but check subtree for timestamps instead of TODO entries. The command offers to archive the subtree if it *does* contain a timestamp, and that timestamp is in the past.

The default archive location is a file in the same directory as the current file, with the name derived by appending ‘`_archive`’ to the current file name. You can also choose what heading to file archived items under, with the possibility to add them to a datetree in a file. For information and examples on how to specify the file and the heading, see the documentation string of the variable `org-archive-location`.

There is also an in-buffer option for setting this variable, for example:

```
#+ARCHIVE: %s_done::
```

If you would like to have a special archive location for a single entry or a (sub)tree, give the entry an ‘ARCHIVE’ property with the location as the value (see Properties and Columns).

When a subtree is moved, it receives a number of special properties that record context information like the file from where the entry came, its outline path the archiving time etc. Configure the variable `org-archive-save-context-info` to adjust the amount of information added.

When `org-archive-subtree-save-file-p` is non-`nil`, save the target archive buffer.

9.2.2 Internal archiving

If you want to just switch off—for agenda views—certain subtrees without moving them to a different file, you can use the ‘ARCHIVE’ tag.

A headline that is marked with the ‘ARCHIVE’ tag (see Tags) stays at its location in the outline tree, but behaves in the following way:

- It does not open when you attempt to do so with a visibility cycling command (see Visibility Cycling). You can force cycling archived subtrees with `C-c C-TAB`, or by setting the option `org-cycle-open-archived-trees`. Also normal outline commands, like `org-show-all`, open archived subtrees.
- During sparse tree construction (see Sparse Trees), matches in archived subtrees are not exposed, unless you configure the option `org-sparse-tree-open-archived-trees`.
- During agenda view construction (see Agenda Views), the content of archived trees is ignored unless you configure the option `org-agenda-skip-archived-trees`, in which case these trees are always included. In the agenda you can press `v a` to get archives temporarily included.
- Archived trees are not exported (see Exporting), only the headline is. Configure the details using the variable `org-export-with-archived-trees`.
- Archived trees are excluded from column view unless the variable `org-columns-skip-archived-trees` is configured to `nil`.

The following commands help manage the ‘ARCHIVE’ tag:

`C-c C-x a` (**`org-toggle-archive-tag`**) ¶
Toggle the archive tag for the current headline. When the tag is set, the headline changes to a shadowed face, and the subtree below it is hidden.

`C-u C-c C-x a` ¶
Check if any direct children of the current headline should be archived. To do this, check each subtree for open TODO entries. If none is found, the command offers to set the ‘ARCHIVE’ tag for the child. If point is *not* on a headline when this command is invoked, check the level 1 trees.

`C-c C-TAB` (**`org-cycle-force-archived`**) ¶
Cycle a tree even if it is tagged with ‘ARCHIVE’.

`C-c C-x A` (**`org-archive-to-archive-sibling`**) ¶

Move the current entry to the *Archive Sibling*. This is a sibling of the entry with the heading 'Archive' and the archive tag. The entry becomes a child of that sibling and in this way retains a lot of its original context, including inherited tags and approximate position in the outline.

---

10 Capture and Attachments

An important part of any organization system is the ability to quickly capture new ideas and tasks, and to associate reference material with them. Org does this using a process called *capture*. It also can store files related to a task (*attachments*) in a special directory. Finally, it can parse RSS feeds for information. To learn how to let external programs (for example a web browser) trigger Org to capture material, see Protocols for External Access.

- Capture
- Attachments
- RSS Feeds

10.1 Capture

Capture lets you quickly store notes with little interruption of your work flow. Org's method for capturing new items is heavily inspired by John Wiegley's excellent Remember package.

10.1.1 Setting up capture

The following customization sets a default target file for notes.

```
(setq org-default-notes-file (concat org-directory "/notes.org"))
```

You may also define a global key for capturing new material (see Activation).

10.1.2 Using capture

`M-x org-capture` (**`org-capture`**) ¶
Display the capture templates menu. If you have templates defined (see Capture templates), it offers these templates for selection or use a new Org outline node as the default template. It inserts the template into the target file and switch to an indirect buffer narrowed to this new node. You may then insert the information you want.

`C-c C-c` (**`org-capture-finalize`**) ¶
Once you have finished entering information into the capture buffer, `C-c C-c` returns you to the window configuration before the capture process, so that you can resume your work without further distraction. When called with a prefix argument, finalize and then jump to the captured item.

`C-c C-w` (**`org-capture-refile`**) ¶
Finalize the capture process by refiling the note to a different place (see Refile and Copy). Please realize that this is a normal refiling command that will be executed—so point position at the moment you run this command is important. If you have inserted a tree with a parent and children, first move point back to the parent. Any prefix argument given to this command is passed on to the `org-refile` command.

`C-c C-k` (**`org-capture-kill`**) ¶

Abort the capture process and return to the previous state.

You can also call `org-capture` in a special way from the agenda, using the `k c` key combination. With this access, any timestamps inserted by the selected capture template defaults to the date at point in the agenda, rather than to the current date.

To find the locations of the last stored capture, use `org-capture` with prefix commands:

`C-u M-x org-capture`
Visit the target location of a capture template. You get to select the template in the usual way.

`C-u C-u M-x org-capture`
Visit the last stored capture item in its buffer.

You can also jump to the bookmark `org-capture-last-stored`, which is automatically created unless you customize `org-bookmark-names-plist`.

To insert the capture at point in an Org buffer, call `org-capture` with a `C-0` prefix argument.

10.1.3 Capture templates

You can use templates for different types of capture items, and for different target locations. The easiest way to create such templates is through the customize interface.

Customize the variable `org-capture-templates`.

Before we give the formal description of template definitions, let's look at an example. Say you would like to use one template to create general TODO entries, and you want to put these entries under the heading '`Tasks`' in your file '`~/org/gtd.org`'. Also, a date tree in the file '`journal.org`' should capture journal entries. A possible configuration would look like:

```
(setq org-capture-templates
      '(("t" "Todo" entry (file+headline "~/org/gtd.org" "Tasks")
         "* TODO %?\n  %i\n  %a")
        ("j" "Journal" entry (file+datetree "~/org/journal.org")
         "* %?\nEntered on %U\n  %i\n  %a")))
```

If you then press `t` from the capture menu, Org will prepare the template for you like this:

```
* TODO
  [[file:LINK TO WHERE YOU INITIATED CAPTURE]]
```

During expansion of the template, '%a' has been replaced by a link to the location from where you called the capture command. This can be extremely useful for deriving tasks from emails, for example. You fill in the task definition, press `C-c C-c` and Org returns you to the same place where you started the capture process.

To define special keys to capture to a particular template without going through the interactive template selection, you can create your key binding like this:

```
(define-key global-map (kbd "C-c x")
  (lambda () (interactive) (org-capture nil "x")))
```

10.1.3.1 Template elements

Now lets look at the elements of a template definition. Each entry in `org-capture-templates` is a list with the following items:

keys
The keys that selects the template, as a string, characters only, for example ‘`"a"`’, for a template to be selected with a single key, or ‘`"bt"`’ for selection with two keys. When using several keys, keys using the same prefix key must be sequential in the list and preceded by a 2-element entry explaining the prefix key, for example:

```
("b" "Templates for marking stuff to buy")
```

If you do not define a template for the `C` key, this key opens the Customize buffer for this complex variable.

description
A short string describing the template, shown during selection.

type
The type of entry, a symbol. Valid values are:

`entry`
An Org mode node, with a headline. Will be filed as the child of the target entry or as a top-level entry. The target file should be an Org file.

`item`
A plain list item, placed in the first plain list at the target location. Again the target file should be an Org file.

`checkitem`
A checkbox item. This only differs from the plain list item by the default template.

`table-line`
A new line in the first table at the target location. Where exactly the line will be inserted depends on the properties `:prepend` and `:table-line-pos` (see below).

`plain`
Text to be inserted as it is.

target ¶
Specification of where the captured item should be placed. In Org files, targets usually define a node. Entries will become children of this node. Other types will be added to the table or list in the body of this node. Most target specifications contain a file name. If that file name is the empty string, it defaults to `org-default-notes-file`. A file can also be given as a variable or as a function called with no argument. When an absolute path is not specified for a target, it is taken as relative to `org-directory`.

Valid values are:

‘`(file "path/to/file")`’
Text will be placed at the beginning or end of that file.

‘`(id "id of existing org entry")`’

Filing as child of this entry, or in the body of the entry.

‘(file+headline "filename" "node headline")’
Fast configuration if the target heading is unique in the file.

‘(file+olp "filename" "Level 1 heading" "Level 2" ...)’
For non-unique headings, the full path is safer.

‘(file+regexp "filename" "regexp to find location")’
Use a regular expression to position point.

‘(file+olp+datetree "filename" ["Level 1 heading" ...])’
This target[83] creates a heading in a date tree[84] for today’s date. If the optional outline path is given, the tree will be built under the node it is pointing to, instead of at top level. Check out the `:time-prompt` and `:tree-type` properties below for additional options.

‘(file+function "filename" function-finding-location)’
A function to find the right location in the file.

‘(clock)’
File to the entry that is currently being clocked.

‘(here)’
The position of ‘`point`’.

‘(function function-finding-location)’
Most general way: write your own function which both visits the file and moves point to the right location.

template
The template for creating the capture item. If you leave this empty, an appropriate default template will be used. Otherwise this is a string with escape codes, which will be replaced depending on time and context of the capture call. You may also get this template string from a file[85], or dynamically, from a function using either syntax:

```
(file "/path/to/template-file")
(function FUNCTION-RETURNING-THE-TEMPLATE)
```

properties
The rest of the entry is a property list of additional options. Recognized properties are:

`:prepend`
Normally new captured information will be appended at the target location (last child, last table line, last list item, …). Setting this property changes that.

`:immediate-finish`
When set, do not offer to edit the information, just file it away immediately. This makes sense if the template only needs information that can be added automatically.

`:jump-to-captured`
When set, jump to the captured entry when finished.

`:empty-lines`
Set this to the number of lines to insert before and after the new item. Default 0, and the only other common value is 1.

`:empty-lines-after`
Set this to the number of lines that should be inserted after the new item. Overrides `:empty-lines` for the number of lines inserted after.

`:empty-lines-before`
Set this to the number of lines that should be inserted before the new item. Overrides `:empty-lines` for the number lines inserted before.

`:clock-in`
Start the clock in this item.

`:clock-keep`
Keep the clock running when filing the captured entry.

`:clock-resume`
If starting the capture interrupted a clock, restart that clock when finished with the capture. Note that `:clock-keep` has precedence over `:clock-resume`. When setting both to non-`nil`, the current clock will run and the previous one will not be resumed.

`:time-prompt`
Prompt for a date/time to be used for date/week trees and when filling the template. Without this property, capture uses the current date and time. Even if this property has not been set, you can force the same behavior by calling `org-capture` with a `C-1` prefix argument.

`:tree-type`
Use `week` to make a week tree instead of the month-day tree, i.e., place the headings for each day under a heading with the current ISO week. Use `month` to group entries by month only. Default is to group entries by day.

`:unnarrowed`
Do not narrow the target buffer, simply show the full buffer. Default is to narrow it so that you only see the new material.

`:table-line-pos`
Specification of the location in the table where the new line should be inserted. It should be a string like ‘`II-3`’ meaning that the new line should become the third line before the second horizontal separator line.

`:kill-buffer`
If the target file was not yet visited when capture was invoked, kill the buffer again after capture is completed.

`:no-save`
Do not save the target file after finishing the capture.

`:refile-targets`
Temporarily set `org-refile-targets` to the value of this property.

`:hook`

A nullary function or list of nullary functions run before `org-capture-mode-hook` when the template is selected.

`:prepare-finalize`
A nullary function or list of nullary functions run before `org-capture-prepare-finalize-hook` when the template is selected.

`:before-finalize`
A nullary function or list of nullary functions run before `org-capture-before-finalize-hook` when the template is selected.

`:after-finalize`
A nullary function or list of nullary functions run before `org-capture-after-finalize-hook` when the template is selected.

10.1.3.2 Template expansion

In the template itself, special "%-escapes"[86] allow dynamic insertion of content. The templates are expanded in the order given here:

'`%[FILE]`'
Insert the contents of the file given by *FILE*.

'`%(EXP)`'
Evaluate Elisp expression *EXP* and replace it with the result. The *EXP* form must return a string. Only placeholders pre-existing within the template, or introduced with '`%[file]`', are expanded this way. Since this happens after expanding non-interactive "%-escapes", those can be used to fill the expression.

'`%<FORMAT>`'
The result of format-time-string on the *FORMAT* specification.

'`%t`'
Timestamp, date only.

'`%T`'
Timestamp, with date and time.

'`%u`', '`%U`'
Like '`%t`', '`%T`' above, but inactive timestamps.

'`%i`'
Initial content, the region when capture is called while the region is active. If there is text before '`%i`' on the same line, such as indentation, and '`%i`' is not inside a '`%(exp)`' form, that prefix is added before every line in the inserted text.

'`%a`'
Annotation, normally the link created with `org-store-link`.

'`%A`'
Like '`%a`', but prompt for the description part.

‘%l’
Like ‘%a’, but only insert the literal link.

‘%L’
Like ‘%l’, but without brackets (the link content itself).

‘%c’
Current kill ring head.

‘%x’
Content of the X clipboard.

‘%k’
Title of the currently clocked task.

‘%K’
Link to the currently clocked task.

‘%n’
User name (taken from `user-full-name`).

‘%f’
File visited by current buffer when org-capture was called.

‘%F’
Full path of the file or directory visited by current buffer.

‘%:keyword’
Specific information for certain link types, see below.

‘%^g’
Prompt for tags, with completion on tags in target file.

‘%^G’
Prompt for tags, with completion all tags in all agenda files.

‘%^t’
Like ‘%t’, but prompt for date. Similarly ‘%^T’, ‘%^u’, ‘%^U’. You may define a prompt like ‘%^{Birthday}t’.

‘%^C’
Interactive selection of which kill or clip to use.

‘%^L’
Like ‘%^C’, but insert as link.

‘%^{PROP}p’
Prompt the user for a value for property *PROP*. You may specify a default value with ‘%^{PROP|default}’.

‘%^{PROMPT}X’, X is one of g,G,t,T,u,U,C,L

Prompt the user as in ‘%^X’, but use the custom prompt string. You may specify a default value and completions with ‘%^{PROMPT|default|completion1|completion2|completion3...}X’.

‘%^{PROMPT}’
Prompt the user for a string and replace this sequence with it. You may specify a default value and a completion table with ‘%^{prompt|default|completion2|completion3...}’. The arrow keys access a prompt-specific history.

‘%\N’
Insert the text entered at the *N*th ‘%^{PROMPT}’, where *N* is a number, starting from 1.

‘%?’
After completing the template, position point here.

For specific link types, the following keywords are defined[87]:

Link type	Available keywords
bbdb	‘%:name’, ‘%:company’
irc	‘%:server’, ‘%:port’, ‘%:nick’
mh, rmail	‘%:type’, ‘%:subject’, ‘%:message-id’
	‘%:from’, ‘%:fromname’, ‘%:fromaddress’
	‘%:to’, ‘%:toname’, ‘%:toaddress’
	‘%:date’ (message date header field)
	‘%:date-timestamp’ (date as active timestamp)
	‘%:date-timestamp-inactive’ (date as inactive timestamp)
	‘%:fromto’ (either “to NAME” or “from NAME”)[88]
gnus	‘%:group’, for messages also all email fields
w3, w3m	‘%:url’
info	‘%:file’, ‘%:node’
calendar	‘%:date’
org-protocol	‘%:link’, ‘%:description’, ‘%:annotation’

10.1.3.3 Templates in contexts

To control whether a capture template should be accessible from a specific context, you can customize `org-capture-templates-contexts`. Let’s say, for example, that you have a capture template “p” for storing Gnus emails containing patches. Then you would configure this option like this:

```
(setq org-capture-templates-contexts
      '(("p" ((in-mode . "message-mode")))))
```

You can also tell that the command key `p` should refer to another template. In that case, add this command key like this:

```
(setq org-capture-templates-contexts
      '(("p" "q" ((in-mode . "message-mode")))))
```

See the docstring of the variable for more information.

10.2 Attachments

It is often useful to associate reference material with an outline node. Small chunks of plain text can simply be stored in the subtree of a project. Hyperlinks (see Hyperlinks) can establish associations with files that live elsewhere on a local, or even remote, computer, like emails or source code files belonging to a project.

Another method is *attachments*, which are files located in a directory belonging to an outline node. Org uses directories either named by a unique ID of each entry, or by a 'DIR' property.

10.2.1 Attachment defaults and dispatcher

By default, Org attach uses ID properties when adding attachments to outline nodes. This makes working with attachments fully automated. There is no decision needed for folder-name or location. ID-based directories are by default located in the 'data/' directory, which lives in the same directory where your Org file lives[89].

When attachments are made using `org-attach` a default tag 'ATTACH' is added to the node that gets the attachments.

For more control over the setup, see Attachment options.

The following commands deal with attachments:

C-c C-a (**org-attach**) ¶
The dispatcher for commands related to the attachment system. After these keys, a list of commands is displayed and you must press an additional key to select a command:

a (**org-attach-attach**) ¶
Select a file and move it into the task's attachment directory. The file is copied, moved, or linked, depending on `org-attach-method`. Note that hard links are not supported on all systems.

c/**m**/**l** ¶
Attach a file using the copy/move/link method. Note that hard links are not supported on all systems.

b (**org-attach-buffer**) ¶
Select a buffer and save it as a file in the task's attachment directory.

n (**org-attach-new**) ¶
Create a new attachment as an Emacs buffer.

z (**org-attach-sync**) ¶
Synchronize the current task with its attachment directory, in case you added attachments yourself.

o (**org-attach-open**) ¶
Open current task's attachment. If there is more than one, prompt for a file name first. Opening follows the rules set by `org-file-apps`. For more details, see the information on following hyperlinks (see Handling Links).

O (**org-attach-open-in-emacs**) ¶
Also open the attachment, but force opening the file in Emacs.

f (**org-attach-reveal**) ¶
Open the current task's attachment directory.

F (**org-attach-reveal-in-emacs**) ¶
Also open the directory, but force using Dired in Emacs.

d (**org-attach-delete-one**) ¶
Select and delete a single attachment.

D (**org-attach-delete-all**) ¶
Delete all of a task's attachments. A safer way is to open the directory in Dired and delete from there.

s (**org-attach-set-directory**) ¶
Set a specific directory as the entry's attachment directory. This works by putting the directory path into the 'DIR' property.

S (**org-attach-unset-directory**) ¶
Remove the attachment directory. This command removes the 'DIR' property and asks the user to either move content inside that folder, if an 'ID' property is set, delete the content, or to leave the attachment directory as is but no longer attached to the outline node.

10.2.2 Attachment options

There are a couple of options for attachments that are worth mentioning.

org-attach-id-dir ¶
The directory where attachments are stored when 'ID' is used as method.

org-attach-dir-relative ¶
When setting the 'DIR' property on a node using `C-c C-a s` (`org-attach-set-directory`), absolute links are entered by default. This option changes that to relative links.

org-attach-use-inheritance ¶
By default folders attached to an outline node are inherited from parents according to `org-use-property-inheritance`. If one instead want to set inheritance specifically for Org attach that can be done using `org-attach-use-inheritance`. Inheriting documents through the node hierarchy makes a lot of sense in most cases. Especially when using attachment links (see Attachment links). The following example shows one use case for attachment inheritance:

```
* Chapter A ...
  :PROPERTIES:
  :DIR: Chapter A/
  :END:
** Introduction
Some text

#+NAME: Image 1
[[attachment:image 1.jpg]]
```

Without inheritance one would not be able to resolve the link to '`image 1.jpg`', since the link is inside a sub-heading to '`Chapter A`'.

Inheritance works the same way for both '`ID`' and '`DIR`' property. If both properties are defined on the same headline then '`DIR`' takes precedence. This is also true if inheritance is enabled. If '`DIR`' is inherited from a parent node in the outline, that property still takes precedence over an '`ID`' property defined on the node itself.

`org-attach-method` ¶
When attaching files using the dispatcher `C-c C-a` it defaults to copying files. The behavior can be changed by customizing `org-attach-method`. Options are Copy, Move/Rename, Hard link or Symbolic link.

`org-attach-preferred-new-method` ¶
This customization lets you choose the default way to attach to nodes without existing '`ID`' and '`DIR`' property. It defaults to `id` but can also be set to `dir`, `ask` or `nil`.

`org-attach-archive-delete` ¶
Configure this to determine if attachments should be deleted or not when a subtree that has attachments is archived.

`org-attach-auto-tag` ¶
When attaching files to a heading it will be assigned a tag according to what is set here.

`org-attach-id-to-path-function-list` ¶
When '`ID`' is used for attachments, the ID is parsed into a part of a directory-path. See `org-attach-id-uuid-folder-format` for the default function. Define a new one and add it as first element in `org-attach-id-to-path-function-list` if you want the folder structure in any other way. All functions in this list will be tried when resolving existing ID's into paths, to maintain backward compatibility with existing folders in your system.

`org-attach-store-link-p` ¶
Stores a link to the file that is being attached. The link is stored in `org-stored-links` for later insertion with `C-c C-l` (see Handling Links). Depending on what option is set in `org-attach-store-link-p`, the link is stored to either the original location as a file link, the attachment location as an attachment link or to the attachment location as a file link.

`org-attach-commands` ¶
List of all commands used in the attach dispatcher.

`org-attach-expert` ¶

Do not show the splash buffer with the attach dispatcher when `org-attach-expert` is set to non-`nil`.

See customization group ‘`Org Attach`’ if you want to change the default settings.

10.2.3 Attachment links

Attached files and folders can be referenced using attachment links. This makes it easy to refer to the material added to an outline node. Especially if it was attached using the unique ID of the entry!

```
* TODO Some task
  :PROPERTIES:
  :ID:       95d50008-c12e-479f-a4f2-cc0238205319
  :END:
See attached document for more information: [[attachment:info.org]]
```

See External Links for more information about these links.

10.2.4 Automatic version-control with Git

If the directory attached to an outline node is a Git repository, Org can be configured to automatically commit changes to that repository when it sees them.

To make Org mode take care of versioning of attachments for you, add the following to your Emacs config:

```
(require 'org-attach-git)
```

10.2.5 Attach from Dired

It is possible to attach files to a subtree from a Dired buffer. To use this feature, have one window in Dired mode containing the file(s) to be attached and another window with point in the subtree that shall get the attachments. In the Dired window, with point on a file, `M-x org-attach-dired-to-subtree` attaches the file to the subtree using the attachment method set by variable `org-attach-method`. When files are marked in the Dired window then all marked files get attached.

Add the following lines to the Emacs init file to have `C-c C-x a` attach files in Dired buffers.

```
(add-hook 'dired-mode-hook
          (lambda ()
            (define-key dired-mode-map
              (kbd "C-c C-x a")
              #'org-attach-dired-to-subtree)))
```

The following code shows how to bind the previous command with a specific attachment method.

```
(add-hook 'dired-mode-hook
          (lambda ()
            (define-key dired-mode-map (kbd "C-c C-x c")
              (lambda ()
                (interactive)
                (let ((org-attach-method 'cp))
                  (call-interactively #'org-attach-dired-to-subtree))))))
```

11 Agenda Views

Due to the way Org works, TODO items, time-stamped items, and tagged headlines can be scattered throughout a file or even a number of files. To get an overview of open action items, or of events that are important for a particular date, this information must be collected, sorted and displayed in an organized way.

Org can select items based on various criteria and display them in a separate buffer. Six different view types are provided:

- an *agenda* that is like a calendar and shows information for specific dates,
- a *TODO list* that covers all unfinished action items,
- a *match view*, showings headlines based on the tags, properties, and TODO state associated with them,
- a *text search view* that shows all entries from multiple files that contain specified keywords,
- a *stuck projects view* showing projects that currently do not move along, and
- *custom views* that are special searches and combinations of different views.

The extracted information is displayed in a special *agenda buffer*. This buffer is read-only, but provides commands to visit the corresponding locations in the original Org files, and even to edit these files remotely.

By default, the report ignores commented (see Comment Lines) and archived (see Internal archiving) entries. You can override this by setting `org-agenda-skip-comment-trees` and `org-agenda-skip-archived-trees` to `nil`.

Two variables control how the agenda buffer is displayed and whether the window configuration is restored when the agenda exits: `org-agenda-window-setup` and `org-agenda-restore-windows-after-quit`.

11.1 Agenda Files

The information to be shown is normally collected from all *agenda files*, the files listed in the variable `org-agenda-files`[90]. If a directory is part of this list, all files with the extension '`.org`' in this directory are part of the list.

Thus, even if you only work with a single Org file, that file should be put into the list[91]. You can customize `org-agenda-files`, but the easiest way to maintain it is through the following commands

C-c [(**org-agenda-file-to-front**) ¶
Add current file to the list of agenda files. The file is added to the front of the list. If it was already in the list, it is moved to the front. With a prefix argument, file is added/moved to the end.

C-c] (**org-remove-file**) ¶
Remove current file from the list of agenda files.

C-' ¶
C-, (**org-cycle-agenda-files**)

Cycle through agenda file list, visiting one file after the other.

`M-x org-switchb` ¶
Command to use an Iswitchb-like interface to switch to and between Org buffers.

The Org menu contains the current list of files and can be used to visit any of them.

If you would like to focus the agenda temporarily on a file not in this list, or on just one file in the list, or even on only a subtree in a file, then this can be done in different ways. For a single agenda command, you may press `<` once or several times in the dispatcher (see The Agenda Dispatcher). To restrict the agenda scope for an extended period, use the following commands:

`C-c C-x <` (`org-agenda-set-restriction-lock`) ¶
Restrict the agenda to the current subtree. If there already is a restriction at point, remove it. When called with a universal prefix argument or with point before the first headline in a file, set the agenda scope to the entire file. This restriction remains in effect until removed with `C-c C-x >`, or by typing either `<` or `>` in the agenda dispatcher. If there is a window displaying an agenda view, the new restriction takes effect immediately.

`C-c C-x >` (`org-agenda-remove-restriction-lock`) ¶
Remove the restriction created by `C-c C-x <`.

When working with Speedbar, you can use the following commands in the Speedbar frame:

`<` (`org-speedbar-set-agenda-restriction`) ¶
Restrict the agenda to the item—either an Org file or a subtree in such a file—at point in the Speedbar frame. If agenda is already restricted there, remove the restriction. If there is a window displaying an agenda view, the new restriction takes effect immediately.

`>` (`org-agenda-remove-restriction-lock`) ¶
Remove the restriction.

11.2 The Agenda Dispatcher

The views are created through a dispatcher, accessible with `M-x org-agenda`, or, better, bound to a global key (see Activation). It displays a menu from which an additional letter is required to execute a command. The dispatcher offers the following default commands:

`a`
Create the calendar-like agenda (see Weekly/daily agenda).

`t`
`T`
Create a list of all TODO items (see The global TODO list).

`m`
`M`
Create a list of headlines matching a given expression (see Matching tags and properties).

`s` ¶
Create a list of entries selected by a boolean expression of keywords and/or regular expressions that must or must not occur in the entry.

`/` ¶
Search for a regular expression in all agenda files and additionally in the files listed in `org-agenda-text-search-extra-files`. This uses the Emacs command `multi-occur`. A prefix argument can be used to specify the number of context lines for each match, the default is 1.

`#`
Create a list of stuck projects (see Stuck projects).

`!`
Configure the list of stuck projects (see Stuck projects).

`<` ¶
Restrict an agenda command to the current buffer[92]. If narrowing is in effect restrict to the narrowed part of the buffer. After pressing `<`, you still need to press the character selecting the command.

`< <` ¶
If there is an active region, restrict the following agenda command to the region. Otherwise, restrict it to the current subtree. After pressing `< <`, you still need to press the character selecting the command.

`*` ¶
Toggle sticky agenda views. By default, Org maintains only a single agenda buffer and rebuilds it each time you change the view, to make sure everything is always up to date. If you switch between views often and the build time bothers you, you can turn on sticky agenda buffers (make this the default by customizing the variable `org-agenda-sticky`). With sticky agendas, the dispatcher only switches to the selected view, you need to update it by hand with `r` or `g`. You can toggle sticky agenda view any time with `org-toggle-sticky-agenda`.

You can also define custom commands that are accessible through the dispatcher, just like the default commands. This includes the possibility to create extended agenda buffers that contain several blocks together, for example the weekly agenda, the global TODO list and a number of special tags matches. See Custom Agenda Views.

11.3 The Built-in Agenda Views

In this section we describe the built-in views.

- Weekly/daily agenda
- The global TODO list
- Matching tags and properties
- Search view
- Stuck projects

Next: The global TODO list, Up: The Built-in Agenda Views [Contents][Index]

11.3.1 Weekly/daily agenda

The purpose of the weekly/daily *agenda* is to act like a page of a paper agenda, showing all the tasks for the current week or day.

`M-x org-agenda a` (`org-agenda-list`) ¶
Compile an agenda for the current week from a list of Org files. The agenda shows the entries for each day. With a numeric prefix argument[93]—like `C-u 2 1 M-x org-agenda a`—you may set the number of days to be displayed.

The default number of days displayed in the agenda is set by the variable `org-agenda-span`. This variable can be set to any number of days you want to see by default in the agenda, or to a span name, such a `day`, `week`, `month` or `year`. For weekly agendas, the default is to start on the previous Monday (see `org-agenda-start-on-weekday`). You can also set the start date using a date shift: '`(setq org-agenda-start-day "+10d")`' starts the agenda ten days from today in the future. `org-agenda-start-on-weekday` takes precedence over `org-agenda-start-day` in weekly and bi-weekly agendas.

Remote editing from the agenda buffer means, for example, that you can change the dates of deadlines and appointments from the agenda buffer. The commands available in the Agenda buffer are listed in Commands in the Agenda Buffer.

Calendar/Diary integration

Emacs contains the calendar and diary by Edward M. Reingold. The calendar displays a three-month calendar with holidays from different countries and cultures. The diary allows you to keep track of anniversaries, lunar phases, sunrise/set, recurrent appointments (weekly, monthly) and more. In this way, it is quite complementary to Org. It can be very useful to combine output from Org with the diary.

In order to include entries from the Emacs diary into Org mode's agenda, you only need to customize the variable

```
(setq org-agenda-include-diary t)
```

After that, everything happens automatically. All diary entries including holidays, anniversaries, etc., are included in the agenda buffer created by Org mode. `SPC`, `TAB`, and `RET` can be used from the agenda buffer to jump to the diary file in order to edit existing diary entries. The `i` command to insert new entries for the current date works in the agenda buffer, as well as the commands `S`, `M`, and `C` to display Sunrise/Sunset times, show lunar phases and to convert to other calendars, respectively. `c` can be used to switch back and forth between calendar and agenda.

If you are using the diary only for expression entries and holidays, it is faster to not use the above setting, but instead to copy or even move the entries into an Org file. Org mode evaluates diary-style expression entries, and does it faster because there is no overhead for first creating the diary display. Note that the expression entries must start at the left margin, no whitespace is allowed before them, as seen in the following segment of an Org file:[94]

```
* Holidays
  :PROPERTIES:
  :CATEGORY: Holiday
  :END:
%%(org-calendar-holiday)   ; special function for holiday names

* Birthdays
  :PROPERTIES:
  :CATEGORY: Ann
  :END:
%%(org-anniversary 1956  5 14) Arthur Dent is %d years old
%%(org-anniversary 1869 10  2) Mahatma Gandhi would be %d years old
```

Anniversaries from BBDB

If you are using the Insidious Big Brother Database to store your contacts, you very likely prefer to store anniversaries in BBDB rather than in a separate Org or diary file. Org supports this and can show BBDB anniversaries as part of the agenda. All you need to do is to add the following to one of your agenda files:

```
* Anniversaries
  :PROPERTIES:
  :CATEGORY: Anniv
  :END:
%%(org-bbdb-anniversaries)
```

You can then go ahead and define anniversaries for a BBDB record. Basically, you need a field named ‘`anniversary`’ for the BBDB record which contains the date in the format ‘`YYYY-MM-DD`’ or ‘`MM-DD`’, followed by a space and the class of the anniversary (‘`birthday`’, ‘`wedding`’, or a format string). If you omit the class, it defaults to ‘`birthday`’. Here are a few examples, the header for the file ‘`ol-bbdb.el`’ contains more detailed information.

```
1973-06-22
06-22
1955-08-02 wedding
2008-04-14 %s released version 6.01 of Org mode, %d years ago
```

After a change to BBDB, or for the first agenda display during an Emacs session, the agenda display suffers a short delay as Org updates its hash with anniversaries. However, from then on things will be very fast, much faster in fact than a long list of ‘`%%(diary-anniversary)`’ entries in an Org or Diary file.

If you would like to see upcoming anniversaries with a bit of forewarning, you can use the following instead:

```
* Anniversaries
  :PROPERTIES:
  :CATEGORY: Anniv
  :END:
%%(org-bbdb-anniversaries-future 3)
```

That will give you three days’ warning: on the anniversary date itself and the two days prior. The argument is optional: if omitted, it defaults to 7.

Appointment reminders

Org can interact with Emacs appointments notification facility. To add the appointments of your agenda files, use the command `org-agenda-to-appt`. This command lets you filter through the list of your appointments and add only those belonging to a specific category or matching a regular expression. It also reads a 'APPT_WARNTIME' property which overrides the value of `appt-message-warning-time` for this appointment. See the docstring for details.

11.3.2 The global TODO list

The global TODO list contains all unfinished TODO items formatted and collected into a single place.

`M-x org-agenda t` (`org-todo-list`) ¶
Show the global TODO list. This collects the TODO items from all agenda files (see Agenda Views) into a single buffer. By default, this lists items with a state that is not a DONE state. The buffer is in Agenda mode, so there are commands to examine and manipulate the TODO entries directly from that buffer (see Commands in the Agenda Buffer).

`M-x org-agenda T` (`org-todo-list`) ¶
Like the above, but allows selection of a specific TODO keyword. You can also do this by specifying a prefix argument to `t`. You are prompted for a keyword, and you may also specify several keywords by separating them with '|' as the boolean OR operator. With a numeric prefix, the Nth keyword in `org-todo-keywords` is selected.

The `r` key in the agenda buffer regenerates it, and you can give a prefix argument to this command to change the selected TODO keyword, for example `3 r`. If you often need a search for a specific keyword, define a custom command for it (see The Agenda Dispatcher).

Matching specific TODO keywords can also be done as part of a tags search (see Tag Searches).

Remote editing of TODO items means that you can change the state of a TODO entry with a single key press. The commands available in the TODO list are described in Commands in the Agenda Buffer.

Normally the global TODO list simply shows all headlines with TODO keywords. This list can become very long. There are two ways to keep it more compact:

- Some people view a TODO item that has been *scheduled* for execution or have a *deadline* (see Timestamps) as no longer *open*. Configure the variables `org-agenda-todo-ignore-scheduled` to exclude some or all scheduled items from the global TODO list, `org-agenda-todo-ignore-deadlines` to exclude some or all items with a deadline set, `org-agenda-todo-ignore-timestamp` to exclude some or all items with an active timestamp other than a DEADLINE or a SCHEDULED timestamp and/or `org-agenda-todo-ignore-with-date` to exclude items with at least one active timestamp.
- TODO items may have sublevels to break up the task into subtasks. In such cases it may be enough to list only the highest level TODO headline and omit the sublevels from the global list. Configure the variable `org-agenda-todo-list-sublevels` to get this behavior.

11.3.3 Matching tags and properties

If headlines in the agenda files are marked with *tags* (see Tags), or have properties (see Properties and Columns), you can select headlines based on this metadata and collect them into an agenda buffer. The match syntax described here also applies when creating sparse trees with `C-c / m`.

`M-x org-agenda m` (`org-tags-view`) ¶
Produce a list of all headlines that match a given set of tags. The command prompts for a selection criterion, which is a boolean logic expression with tags, like '`+work+urgent-withboss`' or '`work|home`' (see Tags). If you often need a specific search, define a custom command for it (see The Agenda Dispatcher).

`M-x org-agenda M` (`org-tags-view`) ¶
Like `m`, but only select headlines that are also TODO items. To exclude scheduled/deadline items, see the variable `org-agenda-tags-todo-honor-ignore-options`. Matching specific TODO keywords together with a tags match is also possible, see Tag Searches.

The commands available in the tags list are described in Commands in the Agenda Buffer.

A search string can use Boolean operators '`&`' for AND and '`|`' for OR. '`&`' binds more strongly than '`|`'. Parentheses are currently not implemented. Each element in the search is either a tag, a regular expression matching tags, or an expression like '`PROPERTY OPERATOR VALUE`' with a comparison operator, accessing a property value. Each element may be preceded by '`-`' to select against it, and '`+`' is syntactic sugar for positive selection. The AND operator '`&`' is optional when '`+`' or '`-`' is present. Here are some examples, using only tags.

'`+work-boss`'
Select headlines tagged '`work`', but discard those also tagged '`boss`'.

'`work|laptop`'
Selects lines tagged '`work`' or '`laptop`'.

'`work|laptop+night`'
Like before, but require the '`laptop`' lines to be tagged also '`night`'.

Instead of a tag, you may also specify a regular expression enclosed in curly braces (see Regular Expressions). For example, '`work+{^boss.*}`' matches headlines that contain the tag '`:work:`' and any tag *starting* with '`boss`'.

Group tags (see Tag Hierarchy) are expanded as regular expressions. E.g., if '`work`' is a group tag for the group '`:work:lab:conf:`', then searching for '`work`' also searches for '`{\(?:work\|lab\|conf\)}`' and searching for '`-work`' searches for all headlines but those with one of the tags in the group (i.e., '`-{\(?:work\|lab\|conf\)}`').

You may also test for properties (see Properties and Columns) at the same time as matching tags. The properties may be real properties, or special properties that represent other metadata (see Special Properties). For example, the property '`TODO`' represents the TODO keyword of the entry. Or, the property '`LEVEL`' represents the level of an entry. So searching

‘`+LEVEL=3+boss-TODO="DONE"`’ lists all level three headlines that have the tag ‘`boss`’ and are *not* marked with the TODO keyword ‘`DONE`’. In buffers with `org-odd-levels-only` set, ‘`LEVEL`’ does not count the number of stars, but ‘`LEVEL=2`’ corresponds to 3 stars etc.

Here are more examples:

‘`work+TODO="WAITING"`’
Select ‘`work`’-tagged TODO lines with the specific TODO keyword ‘`WAITING`’.

‘`work+TODO="WAITING"|home+TODO="WAITING"`’
Waiting tasks both at work and at home.

When matching properties, a number of different operators can be used to test the value of a property. Here is a complex example:

```
+work-boss+PRIORITY="A"+Coffee="unlimited"+Effort<*2
         +With={Sarah\|Denny}+SCHEDULED>="<2008-10-11>"
```

The type of comparison depends on how the comparison value is written:

- If the comparison value is a plain number, a numerical comparison is done, and the allowed operators are ‘<’, ‘=’, ‘>’, ‘<=’, ‘>=’, and ‘<>’. As a synonym for the equality operator ‘=’, there is also ‘==’; ‘`!=`’ and ‘`/=`’ are synonyms of the inequality operator ‘<>’.
- If the comparison value is enclosed in double-quotes, a string comparison is done, and the same operators are allowed.
- If the comparison value is enclosed in double-quotes *and* angular brackets (like ‘`DEADLINE<="<2008-12-24 18:30>"`’), both values are assumed to be date/time specifications in the standard Org way, and the comparison is done accordingly. Valid values also include ‘`"<now>"`’ for now (including time), ‘`"<today>"`’, and ‘`"<tomorrow>"`’ for these days at 0:00 hours, i.e., without a time specification. You can also use strings like ‘`"<+5d>"`’ or ‘`"<-2m>"`’ with units ‘`d`’, ‘`w`’, ‘`m`’, and ‘`y`’ for day, week, month, and year, respectively.
- If the comparison value is enclosed in curly braces, a regexp match is performed, with ‘=’ meaning that the regexp matches the property value, and ‘<>’ meaning that it does not match.
- All operators may be optionally followed by an asterisk ‘`*`’, like in ‘`<*`’, ‘`!=*`’, etc. Such *starred operators* work like their regular, unstarred counterparts except that they match only headlines where the tested property is actually present. This is most useful for search terms that logically exclude results, like the inequality operator.

So the search string in the example finds entries tagged ‘`work`’ but not ‘`boss`’, which also have a priority value ‘`A`’, a ‘`Coffee`’ property with the value ‘`unlimited`’, an ‘`EFFORT`’ property that is numerically smaller than 2, a ‘`With`’ property that is matched by the regular expression ‘`Sarah\|Denny`’, and that are scheduled on or after October 11, 2008.

Note that the test on the ‘`EFFORT`’ property uses operator ‘`<*`’, so that the search result will include only entries that actually have an ‘`EFFORT`’ property defined and with numerical value smaller than 2. With the regular ‘<’ operator, the search would handle entries without an ‘`EFFORT`’ property as having a zero effort and would include them in the result as well.

You can use all characters valid in property names when matching properties. However, you have to quote some characters in property names with backslashes when using them in search strings, namely all characters different from alphanumerics and underscores[95]. For example, to search for all entries having a property ‘`boss-prio`’, ‘`boss:prio`’, or ‘`boss\prio`’, respectively, with value ‘`C`’, use search strings

```
boss\-prio="C"
boss\:prio="C"
boss\\prio="C"
```

You can configure Org mode to use property inheritance during a search, but beware that this can slow down searches considerably. See Property Inheritance, for details.

For backward compatibility, and also for typing speed, there is also a different way to test TODO states in a search. For this, terminate the tags/property part of the search string (which may include several terms connected with ‘`|`’) with a ‘`/`’ and then specify a Boolean expression just for TODO keywords. The syntax is then similar to that for tags, but should be applied with care: for example, a positive selection on several TODO keywords cannot meaningfully be combined with boolean AND. However, *negative selection* combined with AND can be meaningful. To make sure that only lines are checked that actually have any TODO keyword (resulting in a speed-up), use `M-x org-agenda M`, or equivalently start the TODO part after the slash with ‘`!`’. Using `M-x org-agenda M` or ‘`/!`’ does not match TODO keywords in a DONE state. Examples:

‘`work/WAITING`’
Same as ‘`work+TODO="WAITING"`’.

‘`work/!-WAITING-NEXT`’
Select ‘`work`’-tagged TODO lines that are neither ‘`WAITING`’ nor ‘`NEXT`’.

‘`work/!+WAITING|+NEXT`’
Select ‘`work`’-tagged TODO lines that are either ‘`WAITING`’ or ‘`NEXT`’.

11.3.4 Search view

This agenda view is a general text search facility for Org mode entries. It is particularly useful to find notes.

`M-x org-agenda s` (`org-search-view`) ¶
This is a special search that lets you select entries by matching a substring or specific words using a boolean logic.

For example, the search string ‘`computer equipment`’ matches entries that contain ‘`computer equipment`’ as a substring, even if the two words are separated by more space or a line break.

Search view can also search for specific keywords in the entry, using Boolean logic. The search string ‘`+computer +wifi -ethernet -{8\.11[bg]}`’ matches note entries that contain the keywords ‘`computer`’ and ‘`wifi`’, but not the keyword ‘`ethernet`’, and which are also not

matched by the regular expression ‘8\.11[bg]’, meaning to exclude both ‘8.11b’ and ‘8.11g’. The first ‘+’ is necessary to turn on boolean search, other ‘+’ characters are optional. For more details, see the docstring of the command org-search-view.

You can incrementally and conveniently adjust a boolean search from the agenda search view with the following keys

[	Add a positive search word
]	Add a negative search word
{	Add a positive regular expression
}	Add a negative regular expression

Note that in addition to the agenda files, this command also searches the files listed in org-agenda-text-search-extra-files.

11.3.5 Stuck projects

If you are following a system like David Allen’s GTD to organize your work, one of the “duties” you have is a regular review to make sure that all projects move along. A *stuck* project is a project that has no defined next actions, so it never shows up in the TODO lists Org mode produces. During the review, you need to identify such projects and define next actions for them.

M-x org-agenda # (**org-agenda-list-stuck-projects**) ¶
List projects that are stuck.

M-x org-agenda ! ¶
Customize the variable org-stuck-projects to define what a stuck project is and how to find it.

You almost certainly need to configure this view before it works for you. The built-in default assumes that all your projects are level-2 headlines, and that a project is not stuck if it has at least one entry marked with a TODO keyword ‘TODO’ or ‘NEXT’ or ‘NEXTACTION’.

Let’s assume that you, in your own way of using Org mode, identify projects with a tag ‘:PROJECT:’, and that you use a TODO keyword ‘MAYBE’ to indicate a project that should not be considered yet. Let’s further assume that the TODO keyword ‘DONE’ marks finished projects, and that ‘NEXT’ and ‘TODO’ indicate next actions. The tag ‘:@shop:’ indicates shopping and is a next action even without the NEXT tag. Finally, if the project contains the special word ‘IGNORE’ anywhere, it should not be listed either. In this case you would start by identifying eligible projects with a tags/TODO match (see Tag Searches) ‘+PROJECT/-MAYBE-DONE’, and then check for ‘TODO’, ‘NEXT’, ‘@shop’, and ‘IGNORE’ in the subtree to identify projects that are not stuck. The correct customization for this is:

```
(setq org-stuck-projects
      '("+PROJECT/-MAYBE-DONE" ("NEXT" "TODO") ("@shop")
        "\\<IGNORE\\>"))
```

Note that if a project is identified as non-stuck, the subtree of this entry is searched for stuck projects.

11.4 Presentation and Sorting

Before displaying items in an agenda view, Org mode visually prepares the items and sorts them. Each item occupies a single line. The line starts with a *prefix* that contains the *category* (see Categories) of the item and other important information. You can customize in which column tags are displayed through `org-agenda-tags-column`. You can also customize the prefix using the option `org-agenda-prefix-format`. This prefix is followed by a cleaned-up version of the outline headline associated with the item.

- Categories
- Time-of-day specifications
- Sorting of agenda items
- Filtering/limiting agenda items

11.4.1 Categories

The category is a broad label assigned to each agenda item. By default, the category is simply derived from the file name, but you can also specify it with a special line in the buffer, like this:

```
#+CATEGORY: Thesis
```

If you would like to have a special category for a single entry or a (sub)tree, give the entry a 'CATEGORY' property with the special category you want to apply as the value.

The display in the agenda buffer looks best if the category is not longer than 10 characters. You can set up icons for category by customizing the `org-agenda-category-icon-alist` variable.

Next: Sorting of agenda items, Previous: Categories, Up: Presentation and Sorting [Contents] [Index]

11.4.2 Time-of-day specifications

Org mode checks each agenda item for a time-of-day specification. The time can be part of the timestamp that triggered inclusion into the agenda, for example

```
<2005-05-10 Tue 19:00>
```

Time ranges can be specified with two timestamps:

```
<2005-05-10 Tue 20:30>--<2005-05-10 Tue 22:15>
```

In the headline of the entry itself, a time(range)—like '`12:45`' or a '`8:30-1pm`'—may also appear as plain text[96].

If the agenda integrates the Emacs diary (see Weekly/daily agenda), time specifications in diary entries are recognized as well.

For agenda display, Org mode extracts the time and displays it in a standard 24 hour format as part of the prefix. The example times in the previous paragraphs would end up in the agenda like this:

```
 8:30-13:00 Arthur Dent lies in front of the bulldozer
12:45...... Ford Prefect arrives and takes Arthur to the pub
19:00...... The Vogon reads his poem
20:30-22:15 Marvin escorts the Hitchhikers to the bridge
```

If the agenda is in single-day mode, or for the display of today, the timed entries are embedded in a time grid, like

```
 8:00...... ------------------
 8:30-13:00 Arthur Dent lies in front of the bulldozer
10:00...... ------------------
12:00...... ------------------
12:45...... Ford Prefect arrives and takes Arthur to the pub
14:00...... ------------------
16:00...... ------------------
18:00...... ------------------
19:00...... The Vogon reads his poem
20:00...... ------------------
20:30-22:15 Marvin escorts the Hitchhikers to the bridge
```

The time grid can be turned on and off with the variable `org-agenda-use-time-grid`, and can be configured with `org-agenda-time-grid`.

11.4.3 Sorting of agenda items

Before being inserted into a view, the items are sorted. How this is done depends on the type of view.

- For the daily/weekly agenda, the items for each day are sorted. The default order is to first collect all items containing an explicit time-of-day specification. These entries are shown at the beginning of the list, as a *schedule* for the day. After that, items remain grouped in categories, in the sequence given by `org-agenda-files`. Within each category, items are sorted by urgency, which is composed of the base priority (see Priorities; 2000 for priority ‘A’, 1000 for ‘B’, and 0 for ‘C’), plus additional increments for overdue scheduled or deadline items.
- For the TODO list, items remain in the order of categories, but within each category, sorting takes place according to urgency. The urgency used for sorting derives from the priority cookie, with additions depending on how close an item is to its due or scheduled date.
- For tags matches, items are not sorted at all, but just appear in the sequence in which they are found in the agenda files.

Sorting can be customized using the variable `org-agenda-sorting-strategy`, and may also include criteria based on the estimated effort of an entry (see Effort Estimates).

Previous: Sorting of agenda items, Up: Presentation and Sorting [Contents][Index]

11.4.4 Filtering/limiting agenda items

Agenda built-in or custom commands are statically defined. Agenda filters and limits allow flexibly narrowing down the list of agenda entries.

Filters only change the visibility of items, are very fast and are mostly used interactively[97]. You can switch quickly between different filters without having to recreate the agenda. *Limits* on the other hand take effect before the agenda buffer is populated, so they are mostly useful when defined as local variables within custom agenda commands.

Filtering in the agenda

The general filtering command is `org-agenda-filter`, bound to `/`. Before we introduce it, we describe commands for individual filter types. All filtering commands handle prefix arguments in the same way: A single `C-u` prefix negates the filter, so it removes lines selected by the filter. A double prefix adds the new filter condition to the one(s) already in place, so filter elements are accumulated.

`\` (`org-agenda-filter-by-tag`) ¶
Filter the agenda view with respect to a tag. You are prompted for a tag selection letter; `SPC` means any tag at all. Pressing `TAB` at that prompt offers completion to select a tag, including any tags that do not have a selection character. The command then hides all entries that do not contain or inherit this tag. Pressing `+` or `-` at the prompt switches between filtering for and against the next tag. To clear the filter, press `\` twice (once to call the command again, and once at the prompt).

`<` (`org-agenda-filter-by-category`) ¶
Filter by category of the line at point, and show only entries with this category. When called with a prefix argument, hide all entries with the category at point. To clear the filter, call this command again by pressing `<`.

`=` (`org-agenda-filter-by-regexp`) ¶
Filter the agenda view by a regular expression: only show agenda entries matching the regular expression the user entered. To clear the filter, call the command again by pressing `=`.

`_` (`org-agenda-filter-by-effort`) ¶
Filter the agenda view with respect to effort estimates, so select tasks that take the right amount of time. You first need to set up a list of efforts globally, for example

```
(setq org-global-properties
      '(("Effort_ALL". "0 0:10 0:30 1:00 2:00 3:00 4:00")))
```

You can then filter for an effort by first typing an operator, one of `<`, `>` and `=`, and then the one-digit index of an effort estimate in your array of allowed values, where `0` means the 10th value. The filter then restricts to entries with effort smaller-or-equal, equal, or larger-or-equal than the selected value. For application of the operator, entries without a defined effort are treated according to the value of `org-sort-agenda-noeffort-is-high`. To clear the filter, press `_` twice (once to call the command again, and once at the first prompt).

Filter the current agenda view and only display items that fall under the same top-level headline as the current entry. To clear the filter, call this command again by pressing `^`.

`/` (`org-agenda-filter`) ¶
This is the unified interface to four of the five filter methods described above. At the prompt, specify different filter elements in a single string, with full completion support. For example,

```
+work-John+<0:10-/plot/
```

selects entries with category ‘work’ and effort estimates below 10 minutes, and deselects entries with tag ‘John’ or matching the regexp ‘plot’ (see Regular Expressions). You can leave ‘+’ out if that does not lead to ambiguities. The sequence of elements is arbitrary. The filter syntax assumes that there is no overlap between categories and tags. Otherwise, tags take priority. If you reply to the prompt with the empty string, all filtering is removed. If a filter is specified, it replaces all current filters. But if you call the command with a double prefix argument, or if you add an additional ‘+’ (e.g., ‘++work’) to the front of the string, the new filter elements are added to the active ones. A single prefix argument applies the entire filter in a negative sense.

| (org-agenda-filter-remove-all)
Remove all filters in the current agenda view.

Computed tag filtering

If the variable `org-agenda-auto-exclude-function` is set to a user-defined function, that function can select tags that should be used as a tag filter when requested. The function will be called with lower-case versions of all tags represented in the current view. The function should return ‘"-tag"’ if the filter should remove entries with that tag, ‘"+tag"’ if only entries with this tag should be kept, or ‘nil’ if that tag is irrelevant. For example, let’s say you use a ‘Net’ tag to identify tasks which need network access, an ‘Errand’ tag for errands in town, and a ‘Call’ tag for making phone calls. You could auto-exclude these tags based on the availability of the Internet, and outside of business hours, with something like this:

```
(defun my-auto-exclude-fn (tag)
  (when (cond ((string= tag "net")
               (/= 0 (call-process "/sbin/ping" nil nil nil
                                   "-c1" "-q" "-t1" "mail.gnu.org")))
              ((member tag '("errand" "call"))
               (let ((hr (nth 2 (decode-time))))
                 (or (< hr 8) (> hr 21)))))
    (concat "-" tag)))

(setq org-agenda-auto-exclude-function #'my-auto-exclude-fn)
```

You can apply this self-adapting filter by using a triple prefix argument to `org-agenda-filter`, i.e. press `C-u C-u C-u /`, or by pressing `RET` in `org-agenda-filter-by-tag`.

Setting limits for the agenda

Here is a list of options that you can set, either globally, or locally in your custom agenda views (see Custom Agenda Views).

org-agenda-max-entries ¶
Limit the number of entries.

org-agenda-max-effort ¶
Limit the duration of accumulated efforts (as minutes).

org-agenda-max-todos ¶
Limit the number of entries with TODO keywords.

`org-agenda-max-tags` ¶
Limit the number of tagged entries.

When set to a positive integer, each option excludes entries from other categories: for example, '`(setq org-agenda-max-effort 100)`' limits the agenda to 100 minutes of effort and exclude any entry that has no effort property. If you want to include entries with no effort property, use a negative value for `org-agenda-max-effort`. One useful setup is to use `org-agenda-max-entries` locally in a custom command. For example, this custom command displays the next five entries with a 'NEXT' TODO keyword.

```
(setq org-agenda-custom-commands
      '(("n" todo "NEXT"
         ((org-agenda-max-entries 5)))))
```

Once you mark one of these five entry as DONE, rebuilding the agenda will again the next five entries again, including the first entry that was excluded so far.

You can also dynamically set temporary limits, which are lost when rebuilding the agenda:

`~` (`org-agenda-limit-interactively`) ¶
This prompts for the type of limit to apply and its value.

Next: Custom Agenda Views, Previous: Presentation and Sorting, Up: Agenda Views [Contents][Index]

11.5 Commands in the Agenda Buffer

Entries in the agenda buffer are linked back to the Org file or diary file where they originate. You are not allowed to edit the agenda buffer itself, but commands are provided to show and jump to the original entry location, and to edit the Org files "remotely" from the agenda buffer. In this way, all information is stored only once, removing the risk that your agenda and note files may diverge.

Some commands can be executed with mouse clicks on agenda lines. For the other commands, point needs to be in the desired line.

Motion

`n` (`org-agenda-next-line`) ¶
Next line (same as `DOWN` and `C-n`).

`p` (`org-agenda-previous-line`) ¶
Previous line (same as `UP` and `C-p`).

View/Go to Org file

`SPC` or **`mouse-3`** (`org-agenda-show-and-scroll-up`) ¶
Display the original location of the item in another window. With a prefix argument, make sure that drawers stay folded.

`L` (`org-agenda-recenter`) ¶
Display original location and recenter that window.

TAB or **mouse-2** (**org-agenda-goto**) ¶
Go to the original location of the item in another window.

RET (**org-agenda-switch-to**) ¶
Go to the original location of the item and delete other windows.

F (**org-agenda-follow-mode**) ¶
Toggle Follow mode. In Follow mode, as you move point through the agenda buffer, the other window always shows the corresponding location in the Org file. The initial setting for this mode in new agenda buffers can be set with the variable `org-agenda-start-with-follow-mode`.

C-c C-x b (**org-agenda-tree-to-indirect-buffer**) ¶
Display the entire subtree of the current item in an indirect buffer. With a numeric prefix argument N, go up to level N and then take that tree. If N is negative, go up that many levels. With a `C-u` prefix, do not remove the previously used indirect buffer.

C-c C-o (**org-agenda-open-link**) ¶
Follow a link in the entry. This offers a selection of any links in the text belonging to the referenced Org node. If there is only one link, follow it without a selection prompt.

Change display

A ¶
Interactively select another agenda view and append it to the current view.

o ¶
Delete other windows.

v d or short **d** (**org-agenda-day-view**) ¶
Switch to day view. When switching to day view, this setting becomes the default for subsequent agenda refreshes. A numeric prefix argument may be used to jump directly to a specific day of the year. For example, `32 d` jumps to February 1st. When setting day view, a year may be encoded in the prefix argument as well. For example, `200712 d` jumps to January 12, 2007. If such a year specification has only one or two digits, it is expanded into one of the 30 next years or the last 69 years.

v w or short **w** (**org-agenda-week-view**) ¶
Switch to week view. When switching week view, this setting becomes the default for subsequent agenda refreshes. A numeric prefix argument may be used to jump directly to a specific day of the ISO week. For example `9 w` to ISO week number 9. When setting week view, a year may be encoded in the prefix argument as well. For example, `200712 w` jumps to week 12 in 2007. If such a year specification has only one or two digits, it is expanded into one of the 30 next years or the last 69 years.

v m (**org-agenda-month-view**) ¶
Switch to month view. Because month views are slow to create, they do not become the default for subsequent agenda refreshes. A numeric prefix argument may be used to jump directly to a specific day of the month. When setting month view, a year may be encoded in the prefix

argument as well. For example, `200712 m` jumps to December, 2007. If such a year specification has only one or two digits, it is expanded into one of the 30 next years or the last 69 years.

`v y` (`org-agenda-year-view`) ¶
Switch to year view. Because year views are slow to create, they do not become the default for subsequent agenda refreshes. A numeric prefix argument may be used to jump directly to a specific day of the year.

`v SPC` (`org-agenda-reset-view`) ¶
Reset the current view to `org-agenda-span`.

`f` (`org-agenda-later`) ¶
Go forward in time to display the span following the current one. For example, if the display covers a week, switch to the following week. With a prefix argument, repeat that many times.

`b` (`org-agenda-earlier`) ¶
Go backward in time to display earlier dates.

`.` (`org-agenda-goto-today`) ¶
Go to today.

`j` (`org-agenda-goto-date`) ¶
Prompt for a date and go there.

`J` (`org-agenda-clock-goto`) ¶
Go to the currently clocked-in task *in the agenda buffer*.

`D` (`org-agenda-toggle-diary`) ¶
Toggle the inclusion of diary entries. See Weekly/daily agenda.

`v l` or `v L` or short `l` (`org-agenda-log-mode`) ¶
Toggle Logbook mode. In Logbook mode, entries that were marked as done while logging was on (see the variable `org-log-done`) are shown in the agenda, as are entries that have been clocked on that day. You can configure the entry types that should be included in log mode using the variable `org-agenda-log-mode-items`. When called with a `C-u` prefix argument, show all possible logbook entries, including state changes. When called with two prefix arguments `C-u C-u`, show only logging information, nothing else. `v L` is equivalent to `C-u v l`.

`v [` or short `[` (`org-agenda-manipulate-query-add`) ¶
Include inactive timestamps into the current view. Only for weekly/daily agenda.

`v a` (`org-agenda-archives-mode`) ¶
Toggle Archives mode. In Archives mode, trees that are archived (see Internal archiving) are also scanned when producing the agenda. To exit archives mode, press `v a` again. The initial setting for this mode in new agenda buffers can set with the variable `org-agenda-start-with-archives-mode`, which can be set with the same values as `org-agenda-archives-mode`.

`v A` ¶
Toggle Archives mode. Include all archive files as well.

`v R` or short `R` (`org-agenda-clockreport-mode`) ¶
Toggle Clockreport mode. In Clockreport mode, the daily/weekly agenda always shows a table with the clocked times for the time span and file scope covered by the current agenda view. The initial setting for this mode in new agenda buffers can be set with the variable `org-agenda-start-with-clockreport-mode`. By using a prefix argument when toggling this mode (i.e., `C-u R`), the clock table does not show contributions from entries that are hidden by agenda filtering[98]. See also the variables `org-clock-report-include-clocking-task` and `org-agenda-clock-report-header`.

`v c` ¶
Show overlapping clock entries, clocking gaps, and other clocking problems in the current agenda range. You can then visit clocking lines and fix them manually. See the variable `org-agenda-clock-consistency-checks` for information on how to customize the definition of what constituted a clocking problem. To return to normal agenda display, press `l` to exit Logbook mode.

`v E` or short `E` (`org-agenda-entry-text-mode`) ¶
Toggle entry text mode. In entry text mode, a number of lines from the Org outline node referenced by an agenda line are displayed below the line. The maximum number of lines is given by the variable `org-agenda-entry-text-maxlines`. Calling this command with a numeric prefix argument temporarily modifies that number to the prefix value.

`G` (`org-agenda-toggle-time-grid`) ¶
Toggle the time grid on and off. See also the variables `org-agenda-use-time-grid` and `org-agenda-time-grid`.

`r`
`g` **(`org-agenda-redo`)** ¶
Recreate the agenda buffer, for example to reflect the changes after modification of the timestamps of items with `S-LEFT` and `S-RIGHT`. When the buffer is the global TODO list, a prefix argument is interpreted to create a selective list for a specific TODO keyword.

`C-x C-s` or short `s` (`org-save-all-org-buffers`) ¶
Save all Org buffers in the current Emacs session, and also the locations of IDs.

`C-c C-x C-c` (`org-agenda-columns`) ¶
Invoke column view (see Column View) in the agenda buffer. The column view format is taken from the entry at point, or, if there is no entry at point, from the first entry in the agenda view. So whatever the format for that entry would be in the original buffer (taken from a property, from a 'COLUMNS' keyword, or from the default variable `org-columns-default-format`) is used in the agenda.

`C-c C-x >` (`org-agenda-remove-restriction-lock`) ¶
Remove the restriction lock on the agenda, if it is currently restricted to a file or subtree (see Agenda Files).

`M-UP` (`org-agenda-drag-line-backward`) ¶
Drag the line at point backward one line. With a numeric prefix argument, drag backward by that many lines.

Moving agenda lines does not persist after an agenda refresh and does not modify the contributing Org files.

`M-DOWN` (**`org-agenda-drag-line-forward`**) ¶
Drag the line at point forward one line. With a numeric prefix argument, drag forward by that many lines.

Remote editing

`0--9`
Digit argument.

`C-_` (**`org-agenda-undo`**) ¶
Undo a change due to a remote editing command. The change is undone both in the agenda buffer and in the remote buffer.

`t` (**`org-agenda-todo`**) ¶
Change the TODO state of the item, both in the agenda and in the original Org file. A prefix arg is passed through to the `org-todo` command, so for example a `C-u` prefix are will trigger taking a note to document the state change.

`C-S-RIGHT` (**`org-agenda-todo-nextset`**) ¶
Switch to the next set of TODO keywords.

`C-S-LEFT`, **`org-agenda-todo-previousset`** ¶
Switch to the previous set of TODO keywords.

`C-k` (**`org-agenda-kill`**) ¶
Delete the current agenda item along with the entire subtree belonging to it in the original Org file. If the text to be deleted remotely is longer than one line, the kill needs to be confirmed by the user. See variable `org-agenda-confirm-kill`.

`C-c C-w` (**`org-agenda-refile`**) ¶
Refile the entry at point.

`C-c C-x C-a` or short **`a`** (**`org-agenda-archive-default-with-confirmation`**) ¶
Archive the subtree corresponding to the entry at point using the default archiving command set in `org-archive-default-command`. When using the `a` key, confirmation is required.

`C-c C-x a` (**`org-agenda-toggle-archive-tag`**) ¶
Toggle the archive tag (see Internal archiving) for the current headline.

`C-c C-x A` (**`org-agenda-archive-to-archive-sibling`**) ¶
Move the subtree corresponding to the current entry to its *archive sibling*.

`C-c C-x C-s` or short **`$`** (**`org-agenda-archive`**) ¶
Archive the subtree corresponding to the current headline. This means the entry is moved to the configured archive location, most likely a different file.

`T` (**`org-agenda-show-tags`**) ¶
Show all tags associated with the current item. This is useful if you have turned off `org-agenda-show-inherited-tags`, but still want to see all tags of a headline occasionally.

`:` (`org-agenda-set-tags`) ¶
Set tags for the current headline. If there is an active region in the agenda, change a tag for all headings in the region.

`,` (`org-agenda-priority`) ¶
Set the priority for the current item. Org mode prompts for the priority character. If you reply with `SPC`, the priority cookie is removed from the entry.

`+` or `S-UP` (`org-agenda-priority-up`) ¶
Increase the priority of the current item. The priority is changed in the original buffer, but the agenda is not resorted. Use the `r` key for this.

`-` or `S-DOWN` (`org-agenda-priority-down`) ¶
Decrease the priority of the current item.

`C-c C-x e` or short `e` (`org-agenda-set-effort`) ¶
Set the effort property for the current item.

`C-c C-z` or short `z` (`org-agenda-add-note`) ¶
Add a note to the entry. This note is recorded, and then filed to the same location where state change notes are put. Depending on `org-log-into-drawer`, this may be inside a drawer.

`C-c C-a` (`org-attach`) ¶
Dispatcher for all command related to attachments.

`C-c C-s` (`org-agenda-schedule`) ¶
Schedule this item. With a prefix argument, remove the scheduling timestamp

`C-c C-d` (`org-agenda-deadline`) ¶
Set a deadline for this item. With a prefix argument, remove the deadline.

`S-RIGHT` (`org-agenda-do-date-later`) ¶
Change the timestamp associated with the current line by one day into the future. If the date is in the past, the first call to this command moves it to today. With a numeric prefix argument, change it by that many days. For example, `3 6 5 S-RIGHT` changes it by a year. With a `C-u` prefix, change the time by one hour. If you immediately repeat the command, it will continue to change hours even without the prefix argument. With a double `C-u C-u` prefix, do the same for changing minutes. The stamp is changed in the original Org file, but the change is not directly reflected in the agenda buffer. Use `r` or `g` to update the buffer.

`S-LEFT` (`org-agenda-do-date-earlier`) ¶
Change the timestamp associated with the current line by one day into the past.

`>` (`org-agenda-date-prompt`) ¶
Change the timestamp associated with the current line. The key `>` has been chosen, because it is the same as `S-.` on my keyboard.

`I` (`org-agenda-clock-in`) ¶
Start the clock on the current item. If a clock is running already, it is stopped first.

`O` (`org-agenda-clock-out`) ¶

Stop the previously started clock.

X (**org-agenda-clock-cancel**) ¶
Cancel the currently running clock.

J (**org-agenda-clock-goto**) ¶
Jump to the running clock in another window.

k (**org-agenda-capture**) ¶
Like `org-capture`, but use the date at point as the default date for the capture template. See `org-capture-use-agenda-date` to make this the default behavior of `org-capture`.

Bulk remote editing selected entries

m (**org-agenda-bulk-mark**) ¶
Mark the entry at point for bulk action. If there is an active region in the agenda, mark the entries in the region. With numeric prefix argument, mark that many successive entries.

***** (**org-agenda-bulk-mark-all**) ¶
Mark all visible agenda entries for bulk action.

u (**org-agenda-bulk-unmark**) ¶
Unmark entry for bulk action.

U (**org-agenda-bulk-unmark-all**) ¶
Unmark all marked entries for bulk action.

M-m (**org-agenda-bulk-toggle**) ¶
Toggle mark of the entry at point for bulk action.

M-* (**org-agenda-bulk-toggle-all**) ¶
Toggle mark of every entry for bulk action.

% (**org-agenda-bulk-mark-regexp**) ¶
Mark entries matching a regular expression for bulk action.

B (**org-agenda-bulk-action**) ¶
Bulk action: act on all marked entries in the agenda. This prompts for another key to select the action to be applied. The prefix argument to **B** is passed through to the **s** and **d** commands, to bulk-remove these special timestamps. By default, marks are removed after the bulk. If you want them to persist, set `org-agenda-bulk-persistent-marks` to `t` or hit **p** at the prompt.

p
Toggle persistent marks.

$
Archive all selected entries.

A
Archive entries by moving them to their respective archive siblings.

t

Change TODO state. This prompts for a single TODO keyword and changes the state of all selected entries, bypassing blocking and suppressing logging notes—but not timestamps.

`+`
Add a tag to all selected entries.

`-`
Remove a tag from all selected entries.

`s`
Schedule all items to a new date. To shift existing schedule dates by a fixed number of days, use something starting with double plus at the prompt, for example '++8d' or '++2w'.

`d`
Set deadline to a specific date.

`r`
Prompt for a single refile target and move all entries. The entries are no longer in the agenda; refresh (`g`) to bring them back.

`S`
Reschedule randomly into the coming N days. N is prompted for. With a prefix argument (`C-u B S`), scatter only across weekdays.

`f` ¶
Apply a function[99] to marked entries. For example, the function below sets the 'CATEGORY' property of the entries to 'web'.

```
(defun set-category ()
  (interactive "P")
  (let ((marker (or (org-get-at-bol 'org-hd-marker)
                    (org-agenda-error))))
    (org-with-point-at marker
      (org-back-to-heading t)
      (org-set-property "CATEGORY" "web"))))
```

Calendar commands

`c` (`org-agenda-goto-calendar`) ¶
Open the Emacs calendar and go to the date at point in the agenda.

`c` (`org-calendar-goto-agenda`) ¶
When in the calendar, compute and show the Org agenda for the date at point.

`i` (`org-agenda-diary-entry`) ¶
Insert a new entry into the diary, using the date at point and (for block entries) the date at the mark. This adds to the Emacs diary file[100], in a way similar to the `i` command in the calendar. The diary file pops up in another window, where you can add the entry.

If you configure `org-agenda-diary-file` to point to an Org file, Org creates entries in that file instead. Most entries are stored in a date-based outline tree that will later make it easy to archive appointments from previous months/years. The tree is built under an entry with a 'DATE_TREE' property, or else with years as top-level entries. Emacs prompts you for the entry

text—if you specify it, the entry is created in `org-agenda-diary-file` without further interaction. If you directly press `RET` at the prompt without typing text, the target file is shown in another window for you to finish the entry there. See also the `k r` command.

`M` (`org-agenda-phases-of-moon`) ¶
Show the phases of the moon for the three months around current date.

`S` (`org-agenda-sunrise-sunset`) ¶
Show sunrise and sunset times. The geographical location must be set with calendar variables, see the documentation for the Emacs calendar.

`C` (`org-agenda-convert-date`) ¶
Convert the date at point into many other cultural and historic calendars.

`H` (`org-agenda-holidays`) ¶
Show holidays for three months around point date.

Quit and exit

`q` (`org-agenda-quit`) ¶
Quit agenda, remove the agenda buffer.

`x` (`org-agenda-exit`) ¶
Exit agenda, remove the agenda buffer and all buffers loaded by Emacs for the compilation of the agenda. Buffers created by the user to visit Org files are not removed.

11.6 Custom Agenda Views

Custom agenda commands serve two purposes: to store and quickly access frequently used TODO and tags searches, and to create special composite agenda buffers. Custom agenda commands are accessible through the dispatcher (see The Agenda Dispatcher), just like the default commands.

- Storing searches
- Block agenda
- Setting options for custom commands

11.6.1 Storing searches

The first application of custom searches is the definition of keyboard shortcuts for frequently used searches, either creating an agenda buffer, or a sparse tree (the latter covering of course only the current buffer).

Custom commands are configured in the variable `org-agenda-custom-commands`. You can customize this variable, for example by pressing `C` from the agenda dispatcher (see The Agenda Dispatcher). You can also directly set it with Emacs Lisp in the Emacs init file. The following example contains all valid agenda views:

```
(setq org-agenda-custom-commands
      '(("x" agenda)
        ("y" agenda*)
        ("w" todo "WAITING")
        ("W" todo-tree "WAITING")
        ("u" tags "+boss-urgent")
        ("v" tags-todo "+boss-urgent")
        ("U" tags-tree "+boss-urgent")
        ("f" occur-tree "\\<FIXME\\>")
        ("h" . "HOME+Name tags searches") ;description for "h" prefix
        ("hl" tags "+home+Lisa")
        ("hp" tags "+home+Peter")
        ("hk" tags "+home+Kim")))
```

The initial string in each entry defines the keys you have to press after the dispatcher command in order to access the command. Usually this is just a single character, but if you have many similar commands, you can also define two-letter combinations where the first character is the same in several combinations and serves as a prefix key[101]. The second parameter is the search type, followed by the string or regular expression to be used for the matching. The example above will therefore define:

`x`
as a global search for agenda entries planned[102] this week/day.

`y`
as the same search, but only for entries with an hour specification like '`[h]h:mm`'—think of them as appointments.

`w`
as a global search for TODO entries with '`WAITING`' as the TODO keyword.

`W`
as the same search, but only in the current buffer and displaying the results as a sparse tree.

`u`
as a global tags search for headlines tagged '`boss`' but not '`urgent`'.

`v`
The same search, but limiting it to headlines that are also TODO items.

`U`
as the same search, but only in the current buffer and displaying the result as a sparse tree.

`f`
to create a sparse tree (again, current buffer only) with all entries containing the word '`FIXME`'.

`h`
as a prefix command for a '`HOME`' tags search where you have to press an additional key (`l`, `p` or `k`) to select a name (Lisa, Peter, or Kim) as additional tag to match.

Note that `*-tree` agenda views need to be called from an Org buffer as they operate on the current buffer only.

11.6.2 Block agenda

Another possibility is the construction of agenda views that comprise the results of *several* commands, each of which creates a block in the agenda buffer. The available commands include `agenda` for the daily or weekly agenda (as created with `a`) , `alltodo` for the global TODO list (as constructed with `t`), `stuck` for the list of stuck projects (as obtained with `#`) and the matching commands discussed above: `todo`, `tags`, and `tags-todo`.

Here are two examples:

```
(setq org-agenda-custom-commands
      '(("h" "Agenda and Home-related tasks"
         ((agenda "")
          (tags-todo "home")
          (tags "garden")))
        ("o" "Agenda and Office-related tasks"
         ((agenda "")
          (tags-todo "work")
          (tags "office")))))
```

This defines `h` to create a multi-block view for stuff you need to attend to at home. The resulting agenda buffer contains your agenda for the current week, all TODO items that carry the tag ‘`home`’, and also all lines tagged with ‘`garden`’. Finally the command `o` provides a similar view for office tasks.

11.6.3 Setting options for custom commands

Org mode contains a number of variables regulating agenda construction and display. The global variables define the behavior for all agenda commands, including the custom commands. However, if you want to change some settings just for a single custom view, you can do so. Setting options requires inserting a list of variable names and values at the right spot in `org-agenda-custom-commands`. For example:

```
(setq org-agenda-custom-commands
      '(("w" todo "WAITING"
         ((org-agenda-sorting-strategy '(priority-down))
          (org-agenda-prefix-format "  Mixed: ")))
        ("U" tags-tree "+boss-urgent"
         ((org-show-context-detail 'minimal)))
        ("N" search ""
         ((org-agenda-files '("~org/notes.org"))
          (org-agenda-text-search-extra-files nil)))))
```

Now the `w` command sorts the collected entries only by priority, and the prefix format is modified to just say ‘`Mixed:`’ instead of giving the category of the entry. The sparse tags tree of `U` now turns out ultra-compact, because neither the headline hierarchy above the match, nor the headline following the match are shown. The command `N` does a text search limited to only a single file.

For command sets creating a block agenda, `org-agenda-custom-commands` has two separate spots for setting options. You can add options that should be valid for just a single command in the set, and options that should be valid for all commands in the set. The former are just added to the command entry; the latter must come after the list of command entries. Going back to the

block agenda example (see Block agenda), let's change the sorting strategy for the `h` commands to `priority-down`, but let's sort the results for 'garden' tags query in the opposite order, `priority-up`. This would look like this:

```
(setq org-agenda-custom-commands
      '(("h" "Agenda and Home-related tasks"
         ((agenda)
          (tags-todo "home")
          (tags "garden"
                ((org-agenda-sorting-strategy '(priority-up)))))
         ((org-agenda-sorting-strategy '(priority-down))))
        ("o" "Agenda and Office-related tasks"
         ((agenda)
          (tags-todo "work")
          (tags "office")))))
```

As you see, the values and parentheses setting is a little complex. When in doubt, use the customize interface to set this variable—it fully supports its structure. Just one caveat: when setting options in this interface, the *values* are just Lisp expressions. So if the value is a string, you need to add the double-quotes around the value yourself.

To control whether an agenda command should be accessible from a specific context, you can customize `org-agenda-custom-commands-contexts`. Let's say for example that you have an agenda command `o` displaying a view that you only need when reading emails. Then you would configure this option like this:

```
(setq org-agenda-custom-commands-contexts
      '(("o" (in-mode . "message-mode"))))
```

You can also tell that the command key `o` should refer to another command key `r`. In that case, add this command key like this:

```
(setq org-agenda-custom-commands-contexts
      '(("o" "r" (in-mode . "message-mode"))))
```

See the docstring of the variable for more information.

11.7 Exporting Agenda Views

If you are away from your computer, it can be very useful to have a printed version of some agenda views to carry around. Org mode can export custom agenda views as plain text, HTML[103], Postscript, PDF[104], and iCalendar files. If you want to do this only occasionally, use the following command:

`C-x C-w` (`org-agenda-write`) ¶
Write the agenda view to a file.

If you need to export certain agenda views frequently, you can associate any custom agenda command with a list of output file names[105]. Here is an example that first defines custom commands for the agenda and the global TODO list, together with a number of files to which to export them. Then we define two block agenda commands and specify file names for them as well. File names can be relative to the current working directory, or absolute.

```
(setq org-agenda-custom-commands
      '(("X" agenda "" nil ("agenda.html" "agenda.ps"))
        ("Y" alltodo "" nil ("todo.html" "todo.txt" "todo.ps"))
        ("h" "Agenda and Home-related tasks"
         ((agenda "")
          (tags-todo "home")
          (tags "garden"))
         nil
         ("~/views/home.html"))
        ("o" "Agenda and Office-related tasks"
         ((agenda)
          (tags-todo "work")
          (tags "office"))
         nil
         ("~/views/office.ps" "~/calendars/office.ics"))))
```

The extension of the file name determines the type of export. If it is '`.html`', Org mode uses the htmlize package to convert the buffer to HTML and save it to this file name. If the extension is '`.ps`', `ps-print-buffer-with-faces` is used to produce Postscript output. If the extension is '`.ics`', iCalendar export is run export over all files that were used to construct the agenda, and limit the export to entries listed in the agenda. Any other extension produces a plain ASCII file.

The export files are *not* created when you use one of those commands interactively because this might use too much overhead. Instead, there is a special command to produce *all* specified files in one step:

e **(`org-store-agenda-views`)** ¶
Export all agenda views that have export file names associated with them.

You can use the options section of the custom agenda commands to also set options for the export commands. For example:

```
(setq org-agenda-custom-commands
      '(("X" agenda ""
         ((ps-number-of-columns 2)
          (ps-landscape-mode t)
          (org-agenda-prefix-format " [ ] ")
          (org-agenda-with-colors nil)
          (org-agenda-remove-tags t))
         ("theagenda.ps"))))
```

This command sets two options for the Postscript exporter, to make it print in two columns in landscape format—the resulting page can be cut in two and then used in a paper agenda. The remaining settings modify the agenda prefix to omit category and scheduling information, and instead include a checkbox to check off items. We also remove the tags to make the lines compact, and we do not want to use colors for the black-and-white printer. Settings specified in `org-agenda-exporter-settings` also apply, e.g.,

```
(setq org-agenda-exporter-settings
      '((ps-number-of-columns 2)
        (ps-landscape-mode t)
        (org-agenda-add-entry-text-maxlines 5)
        (htmlize-output-type 'css)))
```

but the settings in `org-agenda-custom-commands` take precedence.

From the command line you may also use:

```
emacs -eval (org-batch-store-agenda-views) -kill
```

or, if you need to modify some parameters[106]

```
emacs -eval '(org-batch-store-agenda-views                      \
              org-agenda-span (quote month)                     \
              org-agenda-start-day "2007-11-01"                 \
              org-agenda-include-diary nil                      \
              org-agenda-files (quote ("~/org/project.org")))'  \
      -kill
```

which creates the agenda views restricted to the file '`~/org/project.org`', without diary entries and with a 30-day extent.

You can also extract agenda information in a way that allows further processing by other programs. See Extracting Agenda Information, for more information.

11.8 Using Column View in the Agenda

Column view (see Column View) is normally used to view and edit properties embedded in the hierarchical structure of an Org file. It can be quite useful to use column view also from the agenda, where entries are collected by certain criteria.

`C-c C-x C-c` (`org-agenda-columns`) ¶
Turn on column view in the agenda.

To understand how to use this properly, it is important to realize that the entries in the agenda are no longer in their proper outline environment. This causes the following issues:

1. Org needs to make a decision which columns format to use. Since the entries in the agenda are collected from different files, and different files may have different columns formats, this is a non-trivial problem. Org first checks if `org-overriding-columns-format` is currently set, and if so, takes the format from there. You should set this variable only in the *local settings section* of a custom agenda command (see Custom Agenda Views) to make it valid for that specific agenda view. If no such binding exists, it checks, in sequence, `org-columns-default-format-for-agenda`, the format associated with the first item in the agenda (through a property or a '`#+COLUMNS`' setting in that buffer) and finally `org-columns-default-format`.
2. If any of the columns has a summary type defined (see Column attributes), turning on column view in the agenda visits all relevant agenda files and make sure that the computations of this property are up to date. This is also true for the special '`CLOCKSUM`' property. Org then sums the values displayed in the agenda. In the daily/weekly agenda, the sums cover a single day; in all other views they cover the entire block.
   It is important to realize that the agenda may show the same entry *twice*—for example as scheduled and as a deadline—and it may show two entries from the same hierarchy (for example a *parent* and its *child*). In these cases, the summation in the agenda leads to incorrect results because some values count double.

3. When the column view in the agenda shows the ‘CLOCKSUM’ property, that is always the entire clocked time for this item. So even in the daily/weekly agenda, the clocksum listed in column view may originate from times outside the current view. This has the advantage that you can compare these values with a column listing the planned total effort for a task —one of the major applications for column view in the agenda. If you want information about clocked time in the displayed period use clock table mode (press R in the agenda).
4. When the column view in the agenda shows the ‘CLOCKSUM_T’ property, that is always today’s clocked time for this item. So even in the weekly agenda, the clocksum listed in column view only originates from today. This lets you compare the time you spent on a task for today, with the time already spent—via ‘CLOCKSUM’—and with the planned total effort for it.

12 Markup for Rich Contents

Org is primarily about organizing and searching through your plain-text notes. However, it also provides a lightweight yet robust markup language for rich text formatting and more. For instance, you may want to center or emphasize text. Or you may need to insert a formula or image in your writing. Org offers syntax for all of this and more. Used in conjunction with the export framework (see Exporting), you can author beautiful documents in Org—like the fine manual you are currently reading.

12.1 Paragraphs

Paragraphs are separated by at least one empty line. If you need to enforce a line break within a paragraph, use ‘\\’ at the end of a line.

To preserve the line breaks, indentation and blank lines in a region, but otherwise use normal formatting, you can use this construct, which can also be used to format poetry.

```
#+BEGIN_VERSE
 Great clouds overhead
 Tiny black birds rise and fall
 Snow covers Emacs

    ---AlexSchroeder
#+END_VERSE
```

When quoting a passage from another document, it is customary to format this as a paragraph that is indented on both the left and the right margin. You can include quotations in Org documents like this:

```
#+BEGIN_QUOTE
Everything should be made as simple as possible,
but not any simpler ---Albert Einstein
#+END_QUOTE
```

If you would like to center some text, do it like this:

```
#+BEGIN_CENTER
Everything should be made as simple as possible, \\
but not any simpler
#+END_CENTER
```

12.2 Emphasis and Monospace

You can make words '`*bold*`', '`/italic/`', '`_underlined_`', '`=verbatim=`' and '`~code~`', and, if you must, '`+strike-through+`'. Text in the code and verbatim string is not processed for Org specific syntax; it is exported verbatim. Org provides a single command as entry point for inserting the marker character.

`C-c C-x C-f` (`org-emphasize`) ¶
Prompt for a marker character and insert or change an emphasis. If there is an active region, change that region to a new emphasis. If there is no region, just insert the marker characters and position the cursor between them.

To turn off fontification for marked up text, you can set `org-fontify-emphasized-text` to `nil`. To narrow down the list of the fontified markup syntax, you can customize `org-emphasis-alist`[107].

To hide the emphasis markup characters in your buffers, set `org-hide-emphasis-markers` to `t`.

Sometimes, when marked text also contains the marker character itself, the result may be unsettling. For example,

```
/One may expect this whole sentence to be italicized, but the
following ~user/?variable~ contains =/= character, which effectively
stops emphasis there./
```

You can use zero width space to help Org sorting out the ambiguity. See Escape Character for more details.

12.3 Subscripts and Superscripts

'^' and '_' are used to indicate super- and subscripts. To increase the readability of ASCII text, it is not necessary, but OK, to surround multi-character sub- and superscripts with curly braces. For example

```
The radius of the sun is R_sun = 6.96 x 10^8 m.  On the other hand,
the radius of Alpha Centauri is R_{Alpha Centauri} = 1.28 x R_{sun}.
```

If you write a text where the underscore is often used in a different context, Org's convention to always interpret these as subscripts can get in your way. Configure the variable `org-use-sub-superscripts` and/or `org-export-with-sub-superscripts` to change this convention. For example, when setting these variables to `{}`, 'a_b' is not displayed/exported[108] as a subscript, but 'a_{b}' is.

You can set both `org-use-sub-superscripts` `org-export-with-sub-superscripts` in a file using the export option '`^:`' (see Export Settings). For example, '`#+OPTIONS: ^:{}`' sets the two options to `{}` and limits super- and subscripts to the curly bracket notation.

You can also toggle the visual display of super- and subscripts:

`C-c C-x \` (`org-toggle-pretty-entities`) ¶
This command formats sub- and superscripts in a WYSIWYM way.

Set both `org-pretty-entities` and `org-pretty-entities-include-sub-superscripts` to `t` to start with super- and subscripts *visually* interpreted as specified by the option `org-use-sub-superscripts`.

12.4 Special Symbols

You can use LaTeX-like syntax to insert special symbols—named entities—like '`\alpha`' to indicate the Greek letter, or '`\to`' to indicate an arrow. Completion for these symbols is available, just type '`\`' and maybe a few letters, and press `M-TAB` to see possible completions. If you need such a symbol inside a word, terminate it with a pair of curly brackets. For example

```
Pro tip: Given a circle \Gamma of diameter d, the length of its
circumference is \pi{}d.
```

A large number of entities is provided, with names taken from both HTML and LaTeX; you can comfortably browse the complete list from a dedicated buffer using the command `org-entities-help`. It is also possible to provide your own special symbols in the variable `org-entities-user`.

During export, these symbols are transformed into the native format of the exporter backend. Strings like '`\alpha`' are exported as '`α`' in the HTML output, and as '`\(\alpha\)`' in the LaTeX output. Similarly, '`\nbsp`' becomes '` `' in HTML and '~' in LaTeX.

If you would like to see entities displayed as UTF-8 characters, use the following command[109]:

`C-c C-x \` (`org-toggle-pretty-entities`) ¶
Toggle display of entities as UTF-8 characters. This does not change the buffer content which remains plain ASCII, but it overlays the UTF-8 character for display purposes only.

In addition to regular entities defined above, Org exports in a special way[110] the following commonly used character combinations: '`\-`' is treated as a shy hyphen, '`--`' and '`---`' are converted into dashes, and '`...`' becomes a compact set of dots.

12.5 Embedded LaTeX

Plain ASCII is normally sufficient for almost all note taking. Exceptions include scientific notes, which often require mathematical symbols and the occasional formula. LaTeX[111] is widely used to typeset scientific documents. Org mode supports embedding LaTeX code into its files, because many academics are used to writing and reading LaTeX source code, and because it can be readily processed to produce pretty output for a number of export backends.

- LaTeX fragments
- Previewing LaTeX fragments
- Using CDLaTeX to enter math

Next: Previewing LaTeX fragments, Up: Embedded LaTeX [Contents][Index]

12.5.1 LaTeX fragments

Org mode can contain LaTeX math fragments, and it supports ways to process these for several export backends. When exporting to LaTeX, the code is left as it is. When exporting to HTML, Org can use either MathJax (see Math formatting in HTML export) or transcode the math into

images (see Previewing LaTeX fragments).

LaTeX fragments do not need any special marking at all. The following snippets are identified as LaTeX source code:

- Environments of any kind[112]. The only requirement is that the '`\begin`' statement appears on a new line, preceded by only whitespace.
- Text within the usual LaTeX math delimiters. Prefer '`\(...\)`' for inline fragments. The '`$...$`' alternative has some restrictions and may be a source of confusion. To avoid conflicts with currency specifications, single '$' characters are only recognized as math delimiters if the enclosed text contains at most two line breaks, is directly attached to the '$' characters with no whitespace in between, and if the closing '$' is followed by whitespace or punctuation (but not a dash).
  Sometimes, it may necessary to have a literal dollar symbol even when it is recognized as LaTeX math delimiter. Org provides '`\dollar`' and '`\USD`' entities (see Special Symbols) that are rendered as '$' for such scenarios. Also, see Escape Character.

For example:

```
\begin{equation}                          % arbitrary environments,
x=\sqrt{b}                                % even tables, figures, etc
\end{equation}

If $a^2=b$ and \( b=2 \), then the solution must be
either $$ a=+\sqrt{2} $$ or \[ a=-\sqrt{2} \].
```

LaTeX processing can be configured with the variable `org-export-with-latex`. The default setting is `t` which means MathJax for HTML, and no processing for ASCII and LaTeX backends. You can also set this variable on a per-file basis using one of these lines:

'`#+OPTIONS: tex:t`'	Do the right thing automatically (MathJax)
'`#+OPTIONS: tex:nil`'	Do not process LaTeX fragments at all
'`#+OPTIONS: tex:verbatim`'	Verbatim export, for jsMath or so

Next: Using CDLaTeX to enter math, Previous: LaTeX fragments, Up: Embedded LaTeX [Contents][Index]

12.5.2 Previewing LaTeX fragments

If you have a working LaTeX installation and '`dvipng`', '`dvisvgm`' or '`convert`' installed[113], LaTeX fragments can be processed to produce images of the typeset expressions to be used for inclusion while exporting to HTML (see LaTeX fragments), or for inline previewing within Org mode.

You can customize the variables `org-format-latex-options` and `org-format-latex-header` to influence some aspects of the preview. In particular, the `:scale` (and for HTML export, `:html-scale`) property of the former can be used to adjust the size of the preview images.

`C-c C-x C-l` (`org-latex-preview`) ¶

Produce a preview image of the LaTeX fragment at point and overlay it over the source code. If there is no fragment at point, process all fragments in the current entry—between two headlines.

When called with a single prefix argument, clear all images in the current entry. Two prefix arguments produce a preview image for all fragments in the buffer, while three of them clear all the images in that buffer.

You can turn on the previewing of all LaTeX fragments in a file with

```
#+STARTUP: latexpreview
```

To disable it, simply use

```
#+STARTUP: nolatexpreview
```

12.5.3 Using CDLaTeX to enter math

CDLaTeX mode is a minor mode that is normally used in combination with a major LaTeX mode like AUCTeX in order to speed-up insertion of environments and math templates. Inside Org mode, you can make use of some of the features of CDLaTeX mode. You need to install '`cdlatex.el`' and '`texmathp.el`' (the latter comes also with AUCTeX) from NonGNU ELPA with the Emacs packaging system or alternatively from https://staff.fnwi.uva.nl/c.dominik/Tools/cdlatex/. Do not use CDLaTeX mode itself under Org mode, but use the special version Org CDLaTeX minor mode that comes as part of Org. Turn it on for the current buffer with `M-x org-cdlatex-mode`, or for all Org files with

```
(add-hook 'org-mode-hook #'turn-on-org-cdlatex)
```

When this mode is enabled, the following features are present (for more details see the documentation of CDLaTeX mode):

`C-c {` ¶
Insert an environment template.

`TAB` ¶
The `TAB` key expands the template if point is inside a LaTeX fragment[114]. For example, `TAB` expands '`fr`' to '`\frac{}{}`' and position point correctly inside the first brace. Another `TAB` gets you into the second brace.

Even outside fragments, `TAB` expands environment abbreviations at the beginning of a line. For example, if you write '`equ`' at the beginning of a line and press `TAB`, this abbreviation is expanded to an '`equation`' environment. To get a list of all abbreviations, type `M-x cdlatex-command-help`.

`^` ¶
`_`
Pressing `_` and `^` inside a LaTeX fragment inserts these characters together with a pair of braces. If you use `TAB` to move out of the braces, and if the braces surround only a single character or macro, they are removed again (depending on the variable `cdlatex-simplify-sub-super-scripts`).

`¶
Pressing the backquote followed by a character inserts math macros, also outside LaTeX fragments. If you wait more than 1.5 seconds after the backquote, a help window pops up.

'¶
Pressing the single-quote followed by another character modifies the LaTeX symbol before point with an accent or a font. If you wait more than 1.5 seconds after the single-quote, a help window pops up. Character modification works only inside LaTeX fragments; outside the quote is normal.

12.6 Literal Examples

You can include literal examples that should not be subjected to markup. Such examples are typeset in monospace, so this is well suited for source code and similar examples.

```
#+BEGIN_EXAMPLE
  Some example from a text file.
#+END_EXAMPLE
```

There is one limitation, however. You must insert a comma right before lines starting with either '*', ', *', '#+' or ', #+', as those may be interpreted as outlines nodes or some other special syntax. Org transparently strips these additional commas whenever it accesses the contents of the block.

```
#+BEGIN_EXAMPLE
,* I am no real headline
#+END_EXAMPLE
```

For simplicity when using small examples, you can also start the example lines with a colon followed by a space. There may also be additional whitespace before the colon:

```
Here is an example
   : Some example from a text file.
```

If the example is source code from a programming language, or any other text that can be marked up by Font Lock in Emacs, you can ask for the example to look like the fontified Emacs buffer[115]. This is done with the code block, where you also need to specify the name of the major mode that should be used to fontify the example[116], see Structure Templates for shortcuts to easily insert code blocks.

```
#+BEGIN_SRC emacs-lisp
  (defun org-xor (a b)
    "Exclusive or."
    (if a (not b) b))
#+END_SRC
```

Both in 'example' and in 'src' snippets, you can add a '-n' switch to the '#+BEGIN' line[117], to get the lines of the example numbered. The '-n' takes an optional numeric argument specifying the starting line number of the block. If you use a '+n' switch, the numbering from the previous numbered snippet is continued in the current one. The '+n' switch can also take a numeric argument. This adds the value of the argument to the last line of the previous block to determine the starting line number.

```
#+BEGIN_SRC emacs-lisp -n 20
  ;; This exports with line number 20.
  (message "This is line 21")
#+END_SRC

#+BEGIN_SRC emacs-lisp +n 10
  ;; This is listed as line 31.
  (message "This is line 32")
#+END_SRC
```

In literal examples, Org interprets strings like ‘`(ref:name)`’ as labels, and use them as targets for special hyperlinks like ‘`[[(name)]]`’—i.e., the reference name enclosed in single parentheses. In HTML, hovering the mouse over such a link remote-highlights the corresponding code line[118], which is kind of cool.

You can also add a ‘`-r`’ switch which *removes* the labels from the source code[119]. With the ‘`-n`’ switch, links to these references are labeled by the line numbers from the code listing. Otherwise links use the labels with no parentheses. Here is an example:

```
#+BEGIN_SRC emacs-lisp -n -r
  (save-excursion                (ref:sc)
     (goto-char (point-min))     (ref:jump)
#+END_SRC
In line [[(sc)]] we remember the current position. [[(jump)][Line (jump)]]
jumps to point-min.
```

Source code and examples may be *indented* in order to align nicely with the surrounding text, and in particular with plain list structure (see Plain Lists). By default, Org only retains the relative indentation between lines, e.g., when exporting the contents of the block. However, you can use the ‘`-i`’ switch to also preserve the global indentation, if it does matter. See Editing Source Code.

If the syntax for the label format conflicts with the language syntax, use a ‘`-l`’ switch to change the format, for example

```
#+BEGIN_SRC pascal -n -r -l "((%s))"
```

See also the variable `org-coderef-label-format`.

HTML export also allows examples to be published as text areas (see Text areas in HTML export).

Because the ‘`#+BEGIN`’ … ‘`#+END`’ patterns need to be added so often, a shortcut is provided (see Structure Templates).

`C-c '` (**`org-edit-special`**) ¶
Edit the source code example at point in its native mode. This works by switching to a temporary buffer with the source code. You need to exit by pressing `C-c '` again. The edited version then replaces the old version in the Org buffer. Fixed-width regions—where each line starts with a colon followed by a space—are edited using Artist mode[120] to allow creating ASCII drawings easily. Using this command in an empty line creates a new fixed-width region.

Calling `org-store-link` (see Handling Links) while editing a source code example in a temporary buffer created with `C-c '` prompts for a label. Make sure that it is unique in the current buffer, and insert it with the proper formatting like '`(ref:label)`' at the end of the current line. Then the label is stored as a link '`(label)`', for retrieval with `C-c C-l`.

12.7 Images

An image is a link to an image file[121] that does not have a description part, for example

```
./img/cat.jpg
```

If you wish to define a caption for the image (see Captions) and maybe a label for internal cross-references (see Internal Links), make sure that the link is on a line by itself and precede it with '`CAPTION`' and '`NAME`' keywords as follows:

```
#+CAPTION: This is the caption for the next figure link (or table)
#+NAME:   fig:SED-HR4049
[[./img/a.jpg]]
```

Such images can be displayed within the buffer with the following command:

`C-c C-x C-v` (**`org-toggle-inline-images`**) ¶
Toggle the inline display of linked images. When called with a prefix argument, also display images that do have a link description. You can ask for inline images to be displayed at startup by configuring the variable `org-startup-with-inline-images`[122].

By default, Org mode displays inline images according to their actual width, but no wider than `fill-column` characters.

You can customize the displayed image width using `org-image-actual-width` variable (globally) or '`ORG-IMAGE-ACTUAL-WIDTH`' property (subtree-level)[123]. Their value can be the following:

- (default) Non-`nil`, use the actual width of images when inlining them. If the actual width is too wide, limit it according to `org-image-max-width`.
- When set to a number, use imagemagick (when available) to set the image's width to this value.
- When set to a number in a list, try to get the width from any '`#+ATTR.*`' keyword if it matches a width specification like:

  ```
  #+ATTR_HTML: :width 300px
  ```

  and fall back on that number if none is found.

- When set to `nil`, try to get the width from an '`#+ATTR.*`' keyword and fall back on the original width or `org-image-max-width` if none is found.

`org-image-max-width` limits the maximum displayed image width, but only when the image width is not set explicitly. Possible maximum width can be set to:

- (default) `fill-column`, limit width to `fill-column` number of characters.
- `window`, limit width to current window width.

- integer number, limit width to that specified number of pixels.
- `nil`, do not limit the width.

Org mode can left-align, center or right-align the display of inline images. This setting is controlled (globally) by `org-image-align`. Only standalone images are affected, corresponding to links with no surrounding text in their paragraph except for whitespace. Its value can be the following:

- (default) The symbol `left`, which inserts the image where the link appears in the buffer.
- The symbol `center`, which will preview links centered in the Emacs window.
- The symbol `right`, which will preview links right-aligned in the Emacs window.

Inline image alignment can be specified for each link using the '`#+ATTR.*`' keyword if it matches an alignment specification like:

```
#+ATTR_HTML: :align center
```

Org will use the alignment specification from any '`#+ATTR.*`' keyword, such as '`#+ATTR_HTML`' or '#+ATTR_LATEX', but '`#+ATTR_ORG`' (if present) will override the others. For instance, this link

```
#+ATTR_HTML: :align right
#+ATTR_ORG: :align center
[[/path/to/image/file.png]]
```

will be displayed centered in Emacs but exported right-aligned to HTML.

When '`#+ATTR_ORG`' is not set, inline image alignment is also read from the '`:center`' attribute supported by some export backends (like HTML, LaTeX and Beamer.)

Inline images can also be displayed when cycling the folding state. When custom option `org-cycle-inline-images-display` is set, the visible inline images under subtree will be displayed automatically.

12.8 Captions

You can assign a caption to a specific part of a document by inserting a '`CAPTION`' keyword immediately before it:

```
#+CAPTION: This is the caption for the next table (or link)
| ... | ... |
|-----+-----|
```

Optionally, the caption can take the form:

```
#+CAPTION[Short caption]: Longer caption.
```

Even though images and tables are prominent examples of captioned structures, the same caption mechanism can apply to many others—e.g., LaTeX equations, source code blocks. Depending on the export backend, those may or may not be handled.

12.9 Horizontal Rules

A line consisting of only dashes, and at least 5 of them, is exported as a horizontal line.

13 Exporting

At some point you might want to print your notes, publish them on the web, or share them with people not using Org. Org can convert and export documents to a variety of other formats while retaining as much structure (see Document Structure) and markup (see Markup for Rich Contents) as possible.

The libraries responsible for translating Org files to other formats are called *backends*. Org ships with support for the following backends:

- *ascii* (ASCII format)
- *beamer* (LaTeX Beamer format)
- *html* (HTML format)
- *icalendar* (iCalendar format)
- *latex* (LaTeX format)
- *md* (Markdown format)
- *odt* (OpenDocument Text format)
- *org* (Org format)
- *texinfo* (Texinfo format)
- *man* (Man page format)

Users can install libraries for additional formats from the Emacs packaging system. For easy discovery, these packages have a common naming scheme: `ox-NAME`, where *NAME* is a format. For example, `ox-koma-letter` for *koma-letter* backend. More libraries can be found in the '`org-contrib`' repository (see Installation).

Org only loads backends for the following formats by default: ASCII, HTML, iCalendar, LaTeX, and ODT. Additional backends can be loaded in either of two ways: by configuring the `org-export-backends` variable, or by requiring libraries in the Emacs init file. For example, to load the Markdown backend, add this to your Emacs config:

```
(require 'ox-md)
```

13.1 The Export Dispatcher

The export dispatcher is the main interface for Org's exports. A hierarchical menu presents the currently configured export formats. Options are shown as easy toggle switches on the same screen.

Org also has a minimal prompt interface for the export dispatcher. When the variable `org-export-dispatch-use-expert-ui` is set to a non-`nil` value, Org prompts in the minibuffer. To switch back to the hierarchical menu, press `?`.

`C-c C-e` (**`org-export-dispatch`**) ¶
Invokes the export dispatcher interface. The options show default settings. The `C-u` prefix argument preserves options from the previous export, including any subtree selections.

Org exports the entire buffer by default. If the Org buffer has an active region, then Org exports just that region.

Within the dispatcher interface, the following key combinations can further alter what is exported, and how.

`C-a` ¶
Toggle asynchronous export. Asynchronous export uses an external Emacs process with a specially configured initialization file to complete the exporting process in the background, without tying-up Emacs. This is particularly useful when exporting long documents.

Output from an asynchronous export is saved on the *export stack*. To view this stack, call the export dispatcher with a double `C-u` prefix argument. If already in the export dispatcher menu, `&` displays the stack.

You can make asynchronous export the default by setting `org-export-in-background`.

You can set the initialization file used by the background process by setting `org-export-async-init-file`.

`C-b` ¶
Toggle body-only export. Useful for excluding headers and footers in the export. Affects only those backend formats that have sections like '`<head>...</head>`' in HTML.

To make body-only export the default, customize the variable `org-export-body-only`.

`C-f` ¶
Toggle force-publishing export. Publish functions normally only publish changed files (see [BROKEN LINK: **Triggering Publication]). Forced publishing causes files to be published even if their timestamps do not indicate the file has been changed.

To make forced publishing the default, customize the variable `org-export-force-publishing`. (This is similar to `org-publish-use-timestamps-flag`, but only affects the export dispatcher.)

`C-s` ¶
Toggle subtree export. When turned on, Org exports only the subtree starting from point position at the time the export dispatcher was invoked. Org uses the top heading of this subtree as the document's title. If point is not on a heading, Org uses the nearest enclosing header. If point is in the document preamble, Org signals an error and aborts export.

To make subtree export the default, customize the variable `org-export-initial-scope`.

`C-v` ¶
Toggle visible-only export. This is useful for exporting only certain parts of an Org document by adjusting the visibility of particular headings. See also Sparse Trees.

To make visible-only export the default, customize the variable `org-export-visible-only`.

13.2 Export Settings

Export options can be set: globally with variables; for an individual file by making variables buffer-local with in-buffer settings (see Summary of In-Buffer Settings); by setting individual keywords or specifying them in compact form with the '`OPTIONS`' keyword; or for a tree by

setting properties (see Properties and Columns). Options set at a specific level override options set at a more general level.

In-buffer settings may appear anywhere in the file, either directly or indirectly through a file included using '`#+SETUPFILE: filename or URL`' syntax. Option keyword sets tailored to a particular backend can be inserted from the export dispatcher (see The Export Dispatcher) using the '`Insert template`' command by pressing `#`. To insert keywords individually, a good way to make sure the keyword is correct is to type '#+' and then to use `M-TAB`[126] for completion.

The export keywords available for every backend, and their equivalent global variables, include:

'AUTHOR' ¶
The document author (`user-full-name`).

'CREATOR' ¶
Entity responsible for output generation (`org-export-creator-string`).

'DATE' ¶
A date or a timestamp[127].

'EMAIL' ¶
The email address (`user-mail-address`).

'LANGUAGE' ¶
Language to use for translating certain strings (`org-export-default-language`). With '`#+LANGUAGE: fr`', for example, Org translates '`Table of contents`' to the French '`Table des matières`'[128].

'SELECT_TAGS' ¶
List of tags that will, if present, be selected for export. The default value is `org-export-select-tags` '`("export")`'. When a tree is tagged with '`export`', Org selects that tree and its subtrees for export, ignoring all the other sections that do not possess the '`export`' tag.

When selectively exporting files with '`export`' tags set, Org does not export any text that appears before the first headline.

Note that a file without the '`export`' tags will export all its sections.

To select non-default tags for export, customize `org-export-select-tags` (globally) or add '`#+SELECT_TAGS: tag1 tag2`' to the document.

'EXCLUDE_TAGS' ¶
List of tags that will be excluded from export. The default value is `org-export-exclude-tags` '`("noexport")`'. When a tree is tagged with '`noexport`', Org excludes that tree and its subtrees from export.

Entries tagged with '`noexport`' are unconditionally excluded from the export, even if they have an '`export`' tag. Even if a subtree is not exported, Org executes any code blocks contained there.

To select non-default tags for the exclusion, customize `org-export-exclude-tags` (globally) or add '`#+EXCLUDE_TAGS: tag1 tag2`' to the document.

'TITLE' ¶
Org displays this title. For long titles, use multiple '`#+TITLE`' lines.

'EXPORT_FILE_NAME' ¶
The name of the output file to be generated. Otherwise, Org generates the file name based on the buffer name and the extension based on the backend format.

The '`OPTIONS`' keyword is a compact form. To configure multiple options, use several '`OPTIONS`' lines. '`OPTIONS`' recognizes the following arguments.

' ¶
Toggle smart quotes (`org-export-with-smart-quotes`). Depending on the language used, when activated, Org treats pairs of double quotes as primary quotes, pairs of single quotes as secondary quotes, and single quote marks as apostrophes.

***** ¶
Toggle emphasized text (`org-export-with-emphasize`).

- ¶
Toggle conversion of special strings (`org-export-with-special-strings`).

: ¶
Toggle fixed-width sections (`org-export-with-fixed-width`).

< ¶
Toggle inclusion of time/date active/inactive stamps (`org-export-with-timestamps`).

\n ¶
Toggles whether to preserve line breaks (`org-export-preserve-breaks`).

^ ¶
Toggle TeX-like syntax for sub- and superscripts. If you write '`^:{}`', 'a_{b}' is interpreted, but the simple 'a_b' is left as it is (`org-export-with-sub-superscripts`).

arch ¶
Configure how archived trees are exported. When set to `headline`, the export process skips the contents and processes only the headlines (`org-export-with-archived-trees`).

author ¶
Toggle inclusion of author name into exported file (`org-export-with-author`).

expand-links ¶
Toggle expansion of environment variables in file paths (`org-export-expand-links`).

broken-links ¶
Toggles if Org should continue exporting upon finding a broken internal link. When set to `mark`, Org clearly marks the problem link in the output (`org-export-with-broken-links`).

c ¶
Toggle inclusion of 'CLOCK' keywords (`org-export-with-clocks`).

creator ¶
Toggle inclusion of creator information in the exported file (`org-export-with-creator`).

d ¶
Toggles inclusion of drawers, or list of drawers to include, or list of drawers to exclude (`org-export-with-drawers`).

date ¶
Toggle inclusion of a date into exported file (`org-export-with-date`).

e ¶
Toggle inclusion of entities (`org-export-with-entities`).

email ¶
Toggle inclusion of the author's e-mail into exported file (`org-export-with-email`).

Toggle the inclusion of footnotes (`org-export-with-footnotes`).

Set the number of headline levels for export (`org-export-headline-levels`). Below that level, headlines are treated differently. In most backends, they become list items.

inline ¶
Toggle inclusion of inlinetasks (`org-export-with-inlinetasks`).

num ¶
Toggle section-numbers (`org-export-with-section-numbers`). When set to number N, Org numbers only those headlines at level N or above. Set 'UNNUMBERED' property to non-`nil` to disable numbering of heading and subheadings entirely. Moreover, when the value is '`notoc`' the headline, and all its children, do not appear in the table of contents either (see Table of Contents).

p ¶
Toggle export of planning information (`org-export-with-planning`). "Planning information" comes from lines located right after the headline and contain any combination of these cookies: 'SCHEDULED', 'DEADLINE', or 'CLOSED'.

pri ¶
Toggle inclusion of priority cookies (`org-export-with-priority`).

prop ¶
Toggle inclusion of property drawers, or list the properties to include (`org-export-with-properties`).

stat ¶
Toggle inclusion of statistics cookies (`org-export-with-statistics-cookies`).

tags ¶
Toggle inclusion of tags, may also be `not-in-toc` (`org-export-with-tags`).

tasks ¶
Toggle inclusion of tasks (TODO items); or `nil` to remove all tasks; or `todo` to remove done tasks; or list the keywords to keep (`org-export-with-tasks`).

tex ¶
`nil` does not export; `t` exports; `verbatim` keeps everything in verbatim (`org-export-with-latex`).

timestamp ¶
Toggle inclusion of the creation time in the exported file (`org-export-timestamp-file`).

title ¶
Toggle inclusion of title (`org-export-with-title`).

toc ¶
Toggle inclusion of the table of contents, or set the level limit (`org-export-with-toc`).

todo ¶
Toggle inclusion of TODO keywords into exported text (`org-export-with-todo-keywords`).

| ¶
Toggle inclusion of tables (`org-export-with-tables`).

When exporting subtrees, special node properties can override the above keywords. These properties have an 'EXPORT_' prefix. For example, 'DATE' becomes, 'EXPORT_DATE' when used for a specific subtree. Except for 'SETUPFILE', all other keywords listed above have an 'EXPORT_' equivalent.

If `org-export-allow-bind-keywords` is non-`nil`, Emacs variables can become buffer-local during export by using the 'BIND' keyword. Its syntax is '`#+BIND: variable value`'. This is particularly useful for in-buffer settings that cannot be changed using keywords.

13.3 Table of Contents

The table of contents includes all headlines in the document. Its depth is therefore the same as the headline levels in the file. If you need to use a different depth, or turn it off entirely, set the `org-export-with-toc` variable accordingly. You can achieve the same on a per file basis, using the following '`toc`' item in '`OPTIONS`' keyword:

```
#+OPTIONS: toc:2          (only include two levels in TOC)
#+OPTIONS: toc:nil        (no default TOC at all)
```

Org includes both numbered and unnumbered headlines in the table of contents[129]. If you need to exclude an unnumbered headline, along with all its children, set the '`UNNUMBERED`' property to '`notoc`' value.

```
* Subtree not numbered, not in table of contents either
  :PROPERTIES:
  :UNNUMBERED: notoc
  :END:
```

Org normally inserts the table of contents in front of the exported document. To move the table of contents to a different location, first turn off the default with `org-export-with-toc` variable or with '`#+OPTIONS: toc:nil`'. Then insert '`#+TOC: headlines N`' at the desired location(s).

```
#+OPTIONS: toc:nil
...
#+TOC: headlines 2
```

To adjust the table of contents depth for a specific section of the Org document, append an additional '`local`' parameter. This parameter becomes a relative depth for the current level. The following example inserts a local table of contents, with direct children only.

```
* Section
#+TOC: headlines 1 local
```

Note that for this feature to work properly in LaTeX export, the Org file requires the inclusion of the titletoc package. Because of compatibility issues, titletoc has to be loaded *before* hyperref. Customize the `org-latex-default-packages-alist` variable.

The following example inserts a table of contents that links to the children of the specified target.

```
* Target
  :PROPERTIES:
  :CUSTOM_ID: TargetSection
  :END:
** Heading A
** Heading B
* Another section
#+TOC: headlines 1 :target #TargetSection
```

The '`:target`' attribute is supported in HTML, Markdown, ODT, and ASCII export.

Use the '`TOC`' keyword to generate list of tables—respectively, all listings—with captions.

```
#+TOC: listings
#+TOC: tables
```

Normally Org uses the headline for its entry in the table of contents. But with '`ALT_TITLE`' property, a different entry can be specified for the table of contents.

13.4 Include Files

During export, you can include the content of another file. For example, to include your '`.emacs`' file, you could use:

```
#+INCLUDE: "~/.emacs" src emacs-lisp
```

There are three positional arguments after the include keyword, they are:

1. The file name, this is the sole mandatory argument. Org neither checks for correctness or validates the content in any way.
2. The block name to wrap the file content in. When this is '`example`', '`export`', or '`src`' the content is escaped by `org-escape-code-in-string`. Arbitrary block names may be given, however block names starting with '`:`' must be quoted, i.e. '`":name"`'.

3. The source code language to use for formatting the contents. This is relevant to both ‘`export`’ and ‘`src`’ block types.

If an included file is not specified as having any markup language, Org assumes it be in Org format and proceeds as usual with a few exceptions. Org makes the footnote labels (see Creating Footnotes) in the included file local to that file. The contents of the included file belong to the same structure—headline, item—containing the ‘`INCLUDE`’ keyword. In particular, headlines within the file become children of the current section. That behavior can be changed by providing an additional keyword parameter, ‘`:minlevel`’. It shifts the headlines in the included file to become the lowest level. For example, this syntax makes the included file a sibling of the current top-level headline:

```
#+INCLUDE: "~/my-book/chapter2.org" :minlevel 1
```

Inclusion of only portions of files are specified using ranges parameter with ‘`:lines`’ keyword. The line at the upper end of the range will not be included. The start and/or the end of the range may be omitted to use the obvious defaults.

‘`#+INCLUDE: "~/.emacs" :lines "5-10"`’	Include lines 5 to 10, 10 excluded
‘`#+INCLUDE: "~/.emacs" :lines "-10"`’	Include lines 1 to 10, 10 excluded
‘`#+INCLUDE: "~/.emacs" :lines "10-"`’	Include lines from 10 to EOF

Inclusions may specify a file-link to extract an object matched by `org-link-search`[130] (see Search Options in File Links). The ranges for ‘`:lines`’ keyword are relative to the requested element. Therefore,

```
#+INCLUDE: "./paper.org::*conclusion" :lines "1-20"
```

includes the first 20 lines of the headline named ‘`conclusion`’.

To extract only the contents of the matched object, set ‘`:only-contents`’ property to non-`nil`. This omits any planning lines or property drawers. For example, to include the body of the heading with the custom ID ‘`theory`’, you can use

```
#+INCLUDE: "./paper.org::#theory" :only-contents t
```

The following command allows navigating to the included document:

`C-c '` (**`org-edit-special`**) ¶
Visit the included file at point.

13.5 Macro Replacement

Macros replace text snippets during export. Macros are defined globally in `org-export-global-macros`, or document-wise with the following syntax:

```
#+MACRO: name   replacement text; $1, $2 are arguments
```

which can be referenced using ‘`{{{name(arg1, arg2)}}}`’[131]. For example

```
#+MACRO: poem Rose is $1, violet's $2. Life's ordered: Org assists you.
{{{poem(red,blue)}}}
```

becomes

```
Rose is red, violet's blue.  Life's ordered: Org assists you.
```

As a special case, Org parses any replacement text starting with ‘`(eval`’ as an Emacs Lisp expression and evaluates it accordingly. Within such templates, arguments become strings. Thus, the following macro

```
#+MACRO: gnustamp (eval (concat "GNU/" (capitalize $1)))
```

turns ‘`{{{gnustamp(linux)}}}`’ into ‘`GNU/Linux`’ during export.

Org recognizes macro references in following Org markup areas: paragraphs, headlines, verse blocks, tables cells and lists. Org also recognizes macro references in keywords, such as ‘`CAPTION`’, ‘`TITLE`’, ‘`AUTHOR`’, ‘`DATE`’, and for some backend specific export options.

Org comes with following pre-defined macros:

‘`{{{keyword(NAME)}}}`’ ¶
‘`{{{title}}}`’
‘`{{{author}}}`’
‘`{{{email}}}`’
The ‘`keyword`’ macro collects all values from *NAME* keywords throughout the buffer, separated with white space. ‘`title`’, ‘`author`’ and ‘`email`’ macros are shortcuts for, respectively, ‘`{{{keyword(TITLE)}}}`’, ‘`{{{keyword(AUTHOR)}}}`’ and ‘`{{{keyword(EMAIL)}}}`’.

‘`{{{date}}}`’ ¶
‘`{{{date(FORMAT)}}}`’
This macro refers to the ‘`DATE`’ keyword. *FORMAT* is an optional argument to the ‘`date`’ macro that is used only if ‘`DATE`’ is a single timestamp. *FORMAT* should be a format string understood by `format-time-string`.

‘`{{{time(FORMAT)}}}`’ ¶
‘`{{{modification-time(FORMAT, VC)}}}`’
These macros refer to the document’s date and time of export and date and time of modification. *FORMAT* is a string understood by `format-time-string`. If the second argument to the `modification-time` macro is non-`nil`, Org uses ‘`vc.el`’ to retrieve the document’s modification time from the version control system. Otherwise Org reads the file attributes.

‘`{{{input-file}}}`’ ¶
This macro refers to the filename of the exported file.

‘`{{{property(PROPERTY-NAME)}}}`’ ¶
‘`{{{property(PROPERTY-NAME, SEARCH OPTION)}}}`’
This macro returns the value of property *PROPERTY-NAME* in the current entry. If *SEARCH-OPTION* (see Search Options in File Links) refers to a remote entry, use it instead.

‘`{{{n}}}`’ ¶

‘{{{n(NAME)}}}’
‘{{{n(NAME, ACTION)}}}’
This macro implements custom counters by returning the number of times the macro has been expanded so far while exporting the buffer. You can create more than one counter using different *NAME* values. If *ACTION* is ‘-’, previous value of the counter is held, i.e., the specified counter is not incremented. If the value is a number, the specified counter is set to that value. If it is any other non-empty string, the specified counter is reset to 1. You may leave *NAME* empty to reset the default counter.

Moreover, inline source blocks (see Structure of Code Blocks) use the special ‘`results`’ macro to mark their output. As such, you are advised against re-defining it, unless you know what you are doing.

The surrounding brackets can be made invisible by setting `org-hide-macro-markers` to a non-`nil` value.

Org expands macros at the very beginning of the export process.

13.6 Comment Lines

Lines starting with zero or more whitespace characters followed by one ‘`#`’ and a whitespace are treated as comments and, as such, are not exported.

Likewise, regions surrounded by ‘`#+BEGIN_COMMENT`’ … ‘`#+END_COMMENT`’ are not exported.

Finally, a ‘`COMMENT`’ keyword at the beginning of an entry, but after any other keyword or priority cookie, comments out the entire subtree. In this case, the subtree is not exported and no code block within it is executed either[132]. The command below helps changing the comment status of a headline.

`C-c ;` (`org-toggle-comment`) ¶
Toggle the ‘`COMMENT`’ keyword at the beginning of an entry.

Next: Beamer Export, Previous: Comment Lines, Up: Exporting [Contents][Index]

13.7 ASCII/Latin-1/UTF-8 export

ASCII export produces an output file containing only plain ASCII characters. This is the simplest and most direct text output. It does not contain any Org markup. Latin-1 and UTF-8 export use additional characters and symbols available in these encoding standards. All three of these export formats offer the most basic of text output for maximum portability.

On export, Org fills and justifies text according to the text width set in `org-ascii-text-width`.

Org exports links using a footnote-like style where the descriptive part is in the text and the link is in a note before the next heading. See the variable `org-ascii-links-to-notes` for details.

ASCII export commands

`C-c C-e t a` (`org-ascii-export-to-ascii`) ¶
`C-c C-e t l`
`C-c C-e t u`

Export as an ASCII file with a '`.txt`' extension. For '`myfile.org`', Org exports to '`myfile.txt`', overwriting without warning. For '`myfile.txt`', Org exports to '`myfile.txt.txt`' in order to prevent data loss.

`C-c C-e t A` (`org-ascii-export-to-ascii`) ¶
`C-c C-e t L`
`C-c C-e t U`
Export to a temporary buffer. Does not create a file.

ASCII specific export settings

The ASCII export backend has one extra keyword for customizing ASCII output. Setting this keyword works similar to the general options (see Export Settings).

'SUBTITLE' ¶
The document subtitle. For long subtitles, use multiple '#+SUBTITLE' lines in the Org file. Org prints them on one continuous line, wrapping into multiple lines if necessary.

Header and sectioning structure

Org converts the first three outline levels into headlines for ASCII export. The remaining levels are turned into lists. To change this cut-off point where levels become lists, see Export Settings.

Quoting ASCII text

To insert text within the Org file by the ASCII backend, use one the following constructs, inline, keyword, or export block:

```
Inline text @@ascii:and additional text@@ within a paragraph.

#+ASCII: Some text

#+BEGIN_EXPORT ascii
Org exports text in this block only when using ASCII backend.
#+END_EXPORT
```

ASCII specific attributes

ASCII backend recognizes only one attribute, '`:width`', which specifies the width of a horizontal rule in number of characters. The keyword and syntax for specifying widths is:

```
#+ATTR_ASCII: :width 10
-----
```

ASCII special blocks

Besides '#+BEGIN_CENTER' blocks (see Paragraphs), ASCII backend has these two left and right justification blocks:

```
#+BEGIN_JUSTIFYLEFT
It's just a jump to the left...
#+END_JUSTIFYLEFT

#+BEGIN_JUSTIFYRIGHT
...and then a step to the right.
#+END_JUSTIFYRIGHT
```

13.8 Beamer Export

Org uses Beamer export to convert an Org file tree structure into high-quality interactive slides for presentations. Beamer is a LaTeX document class for creating presentations in PDF, HTML, and other popular display formats.

- Beamer export commands
- Beamer specific export settings
- Frames and Blocks in Beamer
- Beamer specific syntax
- Editing support
- A Beamer example

13.8.1 Beamer export commands

`C-c C-e l b` (`org-beamer-export-to-latex`) ¶
Export as LaTeX file with a '`.tex`' extension. For '`myfile.org`', Org exports to '`myfile.tex`', overwriting without warning.

`C-c C-e l B` (`org-beamer-export-as-latex`) ¶
Export to a temporary buffer. Does not create a file.

`C-c C-e l P` (`org-beamer-export-to-pdf`) ¶
Export as LaTeX file and then convert it to PDF format.

`C-c C-e l O` ¶
Export as LaTeX file, convert it to PDF format, and then open the PDF file.

13.8.2 Beamer specific export settings

Beamer export backend has several additional keywords for customizing Beamer output. These keywords work similar to the general options settings (see Export Settings).

'BEAMER_THEME' ¶
The Beamer layout theme (`org-beamer-theme`). Use square brackets for options. For example:

```
#+BEAMER_THEME: Rochester [height=20pt]
```

'BEAMER_FONT_THEME' ¶
The Beamer font theme.

'BEAMER_INNER_THEME' ¶
The Beamer inner theme.

'BEAMER_OUTER_THEME' ¶
The Beamer outer theme.

Arbitrary lines inserted in the preamble, just before the '`hyperref`' settings.

'DESCRIPTION' ¶

The document description. For long descriptions, use multiple ‘DESCRIPTION’ keywords. By default, ‘hyperref’ inserts ‘DESCRIPTION’ as metadata. Use `org-latex-hyperref-template` to configure document metadata. Use `org-latex-title-command` to configure typesetting of description as part of front matter.

‘KEYWORDS’ ¶
The keywords for defining the contents of the document. Use multiple ‘KEYWORDS’ lines if necessary. By default, ‘hyperref’ inserts ‘KEYWORDS’ as metadata. Use `org-latex-hyperref-template` to configure document metadata. Use `org-latex-title-command` to configure typesetting of keywords as part of front matter.

‘SUBTITLE’ ¶
Document’s subtitle. For typesetting, use `org-beamer-subtitle-format` string. Use `org-latex-hyperref-template` to configure document metadata. Use `org-latex-title-command` to configure typesetting of subtitle as part of front matter.

13.8.3 Frames and Blocks in Beamer

Org transforms heading levels into Beamer’s sectioning elements, frames and blocks. Any Org tree with a not-too-deep-level nesting should in principle be exportable as a Beamer presentation.

- Org headlines become Beamer frames when the heading level in Org is equal to `org-beamer-frame-level` or ‘H’ value in a ‘OPTIONS’ line (see Export Settings).
  Org overrides headlines to frames conversion for the current tree of an Org file if it encounters the ‘BEAMER_ENV’ property set to ‘frame’ or ‘fullframe’. Org ignores whatever `org-beamer-frame-level` happens to be for that headline level in the Org tree. In Beamer terminology, a full frame is a frame without its title.

- Org exports a Beamer frame’s objects as block environments. Org can enforce wrapping in special block types when ‘BEAMER_ENV’ property is set[133]. For valid values see `org-beamer-environments-default`. To add more values, see `org-beamer-environments-extra`.

- If ‘BEAMER_ENV’ is set to ‘appendix’, Org exports the entry as an appendix. When set to ‘note’, Org exports the entry as a note within the frame or between frames, depending on the entry’s heading level. When set to ‘noteNH’, Org exports the entry as a note without its title. When set to ‘againframe’, Org exports the entry with ‘\againframe’ command, which makes setting the ‘BEAMER_REF’ property mandatory because ‘\againframe’ needs frame to resume.
  When ‘ignoreheading’ is set, Org export ignores the entry’s headline but not its content. This is useful for inserting content between frames. It is also useful for properly closing a ‘column’ environment.

  When ‘BEAMER_ACT’ is set for a headline, Org export translates that headline as an overlay or action specification. When enclosed in square brackets, Org export makes the overlay specification a default. Use ‘BEAMER_OPT’ to set any options applicable to the current Beamer frame or block. The Beamer export backend wraps with appropriate angular or square brackets. It also adds the ‘fragile’ option for any code that may require a verbatim block.

  To create a column on the Beamer slide, use the ‘BEAMER_COL’ property for its headline in the Org file. Set the value of ‘BEAMER_COL’ to a decimal number representing the fraction of the total text width. Beamer export uses this value to set the column’s width and fills the column with the contents of the Org entry. If the Org entry has no specific environment defined, Beamer export ignores the heading. If the Org entry has a defined environment, Beamer export uses the heading as title. Behind the scenes, Beamer export automatically handles LaTeX column separations for contiguous headlines. To manually adjust them for any unique configurations needs, use the ‘BEAMER_ENV’ property.

13.8.4 Beamer specific syntax

Since Org’s Beamer export backend is an extension of the LaTeX backend, it recognizes other LaTeX specific syntax—for example, ‘#+LATEX:’ or ‘#+ATTR_LATEX:’. See LaTeX Export, for details.

Beamer export wraps the table of contents generated with ‘toc:t’ ‘OPTION’ keyword in a ‘frame’ environment. Beamer export does not wrap the table of contents generated with ‘TOC’ keyword (see Table of Contents). Use square brackets for specifying options.

```
#+TOC: headlines [currentsection]
```

Insert Beamer-specific code using the following constructs:

```
#+BEAMER: \pause

#+BEGIN_EXPORT beamer
  Only Beamer export backend exports this.
#+END_EXPORT

Text @@beamer:some code@@ within a paragraph.
```

Inline constructs, such as the last one above, are useful for adding overlay specifications to objects with `bold`, `item`, `link`, `radio-target` and `target` types. Enclose the value in angular brackets and place the specification at the beginning of the object as shown in this example:

```
A *@@beamer:<2->@@useful* feature
```

Beamer export recognizes the ‘ATTR_BEAMER’ keyword with the following attributes from Beamer configurations: ‘`:environment`’ for changing local Beamer environment, ‘`:overlay`’ for specifying Beamer overlays in angular or square brackets, and ‘`:options`’ for inserting optional arguments.

```
#+ATTR_BEAMER: :environment nonindentlist
- item 1, not indented
- item 2, not indented
- item 3, not indented

#+ATTR_BEAMER: :overlay <+->
- item 1
- item 2

#+ATTR_BEAMER: :options [Lagrange]
Let $G$ be a finite group, and let $H$ be
a subgroup of $G$.  Then the order of $H$ divides the order of $G$.
```

13.8.5 Editing support

Org Beamer mode is a special minor mode for faster editing of Beamer documents.

```
#+STARTUP: beamer
```

C-c C-b (**org-beamer-select-environment**) ¶
Org Beamer mode provides this key for quicker selections in Beamer normal environments, and for selecting the ‘BEAMER_COL’ property.

13.8.6 A Beamer example

Here is an example of an Org document ready for Beamer export.

```
#+TITLE: Example Presentation
#+AUTHOR: Carsten Dominik
#+OPTIONS: H:2 toc:t num:t
#+LATEX_CLASS: beamer
#+LATEX_CLASS_OPTIONS: [presentation]
#+BEAMER_THEME: Madrid
#+COLUMNS: %45ITEM %10BEAMER_ENV(Env) %10BEAMER_ACT(Act) %4BEAMER_COL(Col)

* This is the first structural section

** Frame 1
*** Thanks to Eric Fraga                                           :B_block:
    :PROPERTIES:
    :BEAMER_COL: 0.48
    :BEAMER_ENV: block
    :END:
    for the first viable Beamer setup in Org
*** Thanks to everyone else                                        :B_block:
    :PROPERTIES:
    :BEAMER_COL: 0.48
    :BEAMER_ACT: <2->
    :BEAMER_ENV: block
    :END:
    for contributing to the discussion
**** This will be formatted as a beamer note                      :B_note:
     :PROPERTIES:
     :BEAMER_env: note
     :END:
** Frame 2 (where we will not use columns)
*** Request
    Please test this stuff!
```

Next: LaTeX Export, Previous: Beamer Export, Up: Exporting [Contents][Index]

13.9 HTML Export

Org mode contains an HTML exporter with extensive HTML formatting compatible with XHTML 1.0 strict standard.

- HTML export commands
- HTML specific export settings
- HTML doctypes
- HTML preamble and postamble
- Exporting to minimal HTML
- Quoting HTML tags
- Headlines in HTML export
- Links in HTML export
- Tables in HTML export
- Images in HTML export
- Math formatting in HTML export
- Text areas in HTML export
- CSS support
- JavaScript supported display of web pages

13.9.1 HTML export commands

`C-c C-e h h` (`org-html-export-to-html`) ¶
Export as HTML file with a '`.html`' extension. For '`myfile.org`', Org exports to '`myfile.html`', overwriting without warning. `C-c C-e h o` exports to HTML and opens it in a web browser.

`C-c C-e h H` (`org-html-export-as-html`) ¶
Exports to a temporary buffer. Does not create a file.

13.9.2 HTML specific export settings

HTML export has a number of keywords, similar to the general options settings described in Export Settings.

'DESCRIPTION' ¶
This is the document's description, which the HTML exporter inserts it as a HTML meta tag in the HTML file. For long descriptions, use multiple '`DESCRIPTION`' lines. The exporter takes care of wrapping the lines properly.

The exporter includes a number of other meta tags, which can be customized by modifying `org-html-meta-tags`.

'HTML_DOCTYPE' ¶
Specify the document type, for example: HTML5 (`org-html-doctype`).

'HTML_CONTAINER' ¶
Specify the HTML container, such as '`div`', for wrapping sections and elements (`org-html-container-element`).

'HTML_LINK_HOME' ¶
The URL for home link (`org-html-link-home`).

'HTML_LINK_UP' ¶
The URL for the up link of exported HTML pages (`org-html-link-up`).

'HTML_MATHJAX' ¶
Options for MathJax (`org-html-mathjax-options`). MathJax is used to typeset LaTeX math in HTML documents. See Math formatting in HTML export, for an example.

'HTML_HEAD' ¶
Arbitrary lines for appending to the HTML document's head (`org-html-head`).

'HTML_HEAD_EXTRA' ¶
More arbitrary lines for appending to the HTML document's head (`org-html-head-extra`).

'KEYWORDS' ¶
Keywords to describe the document's content. HTML exporter inserts these keywords as HTML meta tags. For long keywords, use multiple '`KEYWORDS`' lines.

'LATEX_HEADER' ¶
Arbitrary lines for appending to the preamble; HTML exporter appends when transcoding LaTeX fragments to images (see Math formatting in HTML export).

‘SUBTITLE’ ¶
The document’s subtitle. HTML exporter formats subtitle if document type is ‘HTML5’ and the CSS has a ‘subtitle’ class.

Some of these keywords are explained in more detail in the following sections of the manual.

13.9.3 HTML doctypes

Org can export to various (X)HTML flavors.

Set the `org-html-doctype` variable for different (X)HTML variants. Depending on the variant, the HTML exporter adjusts the syntax of HTML conversion accordingly. Org includes the following ready-made variants:

- `"html4-strict"`
- `"html4-transitional"`
- `"html4-frameset"`
- `"xhtml-strict"`
- `"xhtml-transitional"`
- `"xhtml-frameset"`
- `"xhtml-11"`
- `"html5"`
- `"xhtml5"`

See the variable `org-html-doctype-alist` for details. The default is `"xhtml-strict"`.

Org’s HTML exporter does not by default enable new block elements introduced with the HTML5 standard. To enable them, set `org-html-html5-fancy` to non-`nil`. Or use an ‘OPTIONS’ line in the file to set ‘`html5-fancy`’.

HTML5 documents can have arbitrary ‘#+BEGIN’ … ‘#+END’ blocks. For example:

```
#+BEGIN_aside
  Lorem ipsum
#+END_aside
```

exports to:

```
<aside>
  <p>Lorem ipsum</p>
</aside>
```

while this:

```
#+ATTR_HTML: :controls controls :width 350
#+BEGIN_video
#+HTML: <source src="movie.mp4" type="video/mp4">
#+HTML: <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
#+END_video
```

exports to:

```
<video controls="controls" width="350">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  <p>Your browser does not support the video tag.</p>
</video>
```

When special blocks do not have a corresponding HTML5 element, the HTML exporter reverts to standard translation (see `org-html-html5-elements`). For example, '`#+BEGIN_lederhosen`' exports to `<div class="lederhosen">`.

Special blocks cannot have headlines. For the HTML exporter to wrap the headline and its contents in `<section>` or `<article>` tags, set the '`HTML_CONTAINER`' property for the headline.

13.9.4 HTML preamble and postamble

The HTML exporter has delineations for preamble and postamble. The default value for `org-html-preamble` is `t`, which makes the HTML exporter insert the preamble. See the variable `org-html-preamble-format` for the format string.

Set `org-html-preamble` to a string to override the default format string. If set to a function, the HTML exporter expects the function to return a string upon execution. The HTML exporter inserts this string in the preamble. The HTML exporter does not insert a preamble if `org-html-preamble` is set `nil`.

The above also applies to `org-html-postamble` and `org-html-postamble-format`. In addition, `org-html-postamble` can be set to `auto` (its default value), which makes the HTML exporter build a postamble from looking up author's name, email address, creator's name, and date.

Next: Quoting HTML tags, Previous: HTML preamble and postamble, Up: HTML Export [Contents][Index]

13.9.5 Exporting to minimal HTML

If you want to output a minimal HTML file, with no CSS, no Javascript, no preamble or postamble, here are the variable you would need to set:

```
(setq org-html-head ""
      org-html-head-extra ""
      org-html-head-include-default-style nil
      org-html-head-include-scripts nil
      org-html-preamble nil
      org-html-postamble nil
      org-html-use-infojs nil)
```

13.9.6 Quoting HTML tags

The HTML export backend transforms '<' and '>' to '`<`' and '`>`'. To include raw HTML code in the Org file so the HTML export backend can insert that HTML code in the output, use this inline syntax: '`@@html:...@@`'. For example:

```
@@html:<b>@@bold text@@html:</b>@@
```

For larger raw HTML code blocks, use these HTML export code blocks:

```
#+HTML: Literal HTML code for export

#+BEGIN_EXPORT html
  All lines between these markers are exported literally
#+END_EXPORT
```

Next: Links in HTML export, Previous: Quoting HTML tags, Up: HTML Export [Contents][Index]

13.9.7 Headlines in HTML export

Headlines are exported to '<h1>', '<h2>', etc. Each headline gets the 'id' attribute from 'CUSTOM_ID' property, or a unique generated value, see Internal Links.

When `org-html-self-link-headlines` is set to a non-`nil` value, the text of the headlines is also wrapped in '<a>' tags. These tags have a 'href' attribute making the headlines link to themselves.

13.9.8 Links in HTML export

The HTML export backend transforms Org's internal links (see Internal Links) to equivalent HTML links in the output. The backend similarly handles Org's automatic links created by radio targets (see Radio Targets) similarly. For Org links to external files, the backend transforms the links to *relative* paths.

For Org links to other '.org' files, the backend automatically changes the file extension to '.html' and makes file paths relative. If the '.org' files have an equivalent '.html' version at the same location, then the converted links should work without any further manual intervention. However, to disable this automatic path translation, set `org-html-link-org-files-as-html` to `nil`. When disabled, the HTML export backend substitutes the ID-based links in the HTML output. For more about linking files when publishing to a directory, see Publishing links.

Org files can also have special directives to the HTML export backend. For example, by using '#+ATTR_HTML' lines to specify new format attributes to `<a>` or `<img>` tags. This example shows changing the link's title and style:

```
#+ATTR_HTML: :title The Org mode website :style color:red;
[[https://orgmode.org]]
```

13.9.9 Tables in HTML export

The HTML export backend uses `org-html-table-default-attributes` when exporting Org tables to HTML. By default, the exporter does not draw frames and cell borders. To change for this for a table, use the following lines before the table in the Org file:

```
#+CAPTION: This is a table with lines around and between cells
#+ATTR_HTML: :border 2 :rules all :frame border
```

The HTML export backend preserves column groupings in Org tables (see Column Groups) when exporting to HTML.

Additional options for customizing tables for HTML export.

`org-html-table-align-individual-fields` ¶
Non-`nil` attaches style attributes for alignment to each table field.

`org-html-table-caption-above` ¶
Non-`nil` places caption string at the beginning of the table.

`org-html-table-data-tags` ¶
Opening and ending tags for table data fields.

`org-html-table-default-attributes` ¶
Default attributes and values for table tags.

`org-html-table-header-tags` ¶
Opening and ending tags for table's header fields.

`org-html-table-row-tags` ¶
Opening and ending tags for table rows.

`org-html-table-use-header-tags-for-first-column` ¶
Non-`nil` formats column one in tables with header tags.

13.9.10 Images in HTML export

The HTML export backend has features to convert Org image links to HTML inline images and HTML clickable image links.

When the link in the Org file has no description, the HTML export backend by default in-lines that image. For example: '`[[file:myimg.jpg]]`' is in-lined, while '`[[file:myimg.jpg] [the image]]`' links to the text, '`the image`'. For more details, see the variable `org-html-inline-images`.

On the other hand, if the description part of the Org link is itself another link, such as '`file:`' or '`https:`' URL pointing to an image, the HTML export backend in-lines this image and links to the main image. This Org syntax enables the backend to link low-resolution thumbnail to the high-resolution version of the image, as shown in this example:

```
[[file:highres.jpg][file:thumb.jpg]]
```

To change attributes of in-lined images, use '`#+ATTR_HTML`' lines in the Org file. This example shows realignment to right, and adds `alt` and `title` attributes in support of text viewers and modern web accessibility standards.

```
#+CAPTION: A black cat stalking a spider
#+ATTR_HTML: :alt cat/spider image :title Action! :align right
[[./img/a.jpg]]
```

The HTML export backend copies the '`http`' links from the Org file as-is.

13.9.11 Math formatting in HTML export

LaTeX math snippets (see LaTeX fragments) can be displayed in two different ways on HTML pages. The default is to use the MathJax, which should work out of the box with Org[134135]. Some MathJax display options can be configured via `org-html-mathjax-options`, or in the buffer. For example, with the following settings,

```
#+HTML_MATHJAX: align: left indent: 5em tagside: left
```

equation labels are displayed on the left margin and equations are five em from the left margin.

See the docstring of `org-html-mathjax-options` for all supported variables. The MathJax template can be configure via `org-html-mathjax-template`.

If you prefer, you can also request that LaTeX fragments are processed into small images that will be inserted into the browser page. Before the availability of MathJax, this was the default method for Org files. This method requires that the dvipng program, dvisvgm or ImageMagick suite is available on your system. You can still get this processing with

```
#+OPTIONS: tex:dvipng
```

```
#+OPTIONS: tex:dvisvgm
```

or

```
#+OPTIONS: tex:imagemagick
```

13.9.12 Text areas in HTML export

Before Org mode's Babel, one popular approach to publishing code in HTML was by using '`:textarea`'. The advantage of this approach was that copying and pasting was built into browsers with simple JavaScript commands. Even editing before pasting was made simple.

The HTML export backend can create such text areas. It requires an '`#+ATTR_HTML`' line as shown in the example below with the '`:textarea`' option. This must be followed by either an example or a source code block. Other Org block types do not honor the '`:textarea`' option.

By default, the HTML export backend creates a text area 80 characters wide and height just enough to fit the content. Override these defaults with '`:width`' and '`:height`' options on the '`#+ATTR_HTML`' line.

```
#+ATTR_HTML: :textarea t :width 40
#+BEGIN_EXAMPLE
  (defun org-xor (a b)
     "Exclusive or."
     (if a (not b) b))
#+END_EXAMPLE
```

Next: JavaScript supported display of web pages, Previous: Text areas in HTML export, Up: HTML Export [Contents][Index]

13.9.13 CSS support

You can modify the CSS style definitions for the exported file. The HTML exporter assigns the following special CSS classes[136] to appropriate parts of the document—your style specifications may change these, in addition to any of the standard classes like for headlines, tables, etc.

`p.author`	author information, including email
`p.date`	publishing date
`p.creator`	creator info, about org mode version

`.title`	document title
`.subtitle`	document subtitle
`.todo`	TODO keywords, all not-done states
`.done`	the DONE keywords, all states that count as done
`.WAITING`	each TODO keyword also uses a class named after itself
`.timestamp`	timestamp
`.timestamp-kwd`	keyword associated with a timestamp, like ‘SCHEDULED’
`.timestamp-wrapper`	span around keyword plus timestamp
`.tag`	tag in a headline
`._HOME`	each tag uses itself as a class, “@” replaced by “_”
`.target`	target for links
`.linenr`	the line number in a code example
`.code-highlighted`	for highlighting referenced code lines
`div.outline-N`	div for outline level N (headline plus text)
`div.outline-text-N`	extra div for text at outline level N
`.section-number-N`	section number in headlines, different for each level
`.figure-number`	label like “Figure 1:”
`.table-number`	label like “Table 1:”
`.listing-number`	label like “Listing 1:”
`div.figure`	how to format an in-lined image
`pre.src`	formatted source code
`pre.example`	normal example
`p.verse`	verse paragraph
`div.footnotes`	footnote section headline
`p.footnote`	footnote definition paragraph, containing a footnote
`.footref`	a footnote reference number (always a <sup>)
`.footnum`	footnote number in footnote definition (always <sup>)
`.org-svg`	default class for a linked ‘`.svg`’ image

The HTML export backend includes a compact default style in each exported HTML file. To override the default style with another style, use these keywords in the Org file. They will replace the global defaults the HTML exporter uses.

```
#+HTML_HEAD: <link rel="stylesheet" type="text/css" href="style1.css" />
#+HTML_HEAD_EXTRA: <link rel="alternate stylesheet" type="text/css" href="style2.css" />
```

To just turn off the default style, customize `org-html-head-include-default-style` variable, or use this option line in the Org file.

```
#+OPTIONS: html-style:nil
```

For longer style definitions, either use several 'HTML_HEAD' and 'HTML_HEAD_EXTRA' keywords, or use `<style> ... </style>` blocks around them. Both of these approaches can avoid referring to an external file.

In order to add styles to a subtree, use the 'HTML_CONTAINER_CLASS' property to assign a class to the tree. In order to specify CSS styles for a particular headline, you can use the ID specified in a 'CUSTOM_ID' property. You can also assign a specific class to a headline with the 'HTML_HEADLINE_CLASS' property.

Never change the `org-html-style-default` constant. Instead use other simpler ways of customizing as described above.

13.9.14 JavaScript supported display of web pages

Sebastian Rose has written a JavaScript program especially designed to allow two different ways of viewing HTML files created with Org. One is an *Info*-like mode where each section is displayed separately and navigation can be done with the n and p keys, and some other keys as well, press ? for an overview of the available keys. The second one has a *folding* view, much like Org provides inside Emacs. The script is available at https://orgmode.org/org-info.js and the documentation at https://orgmode.org/worg/code/org-info-js/. The script is hosted on https://orgmode.org, but for reliability, prefer installing it on your own web server.

To use this program, just add this line to the Org file:

```
#+INFOJS_OPT: view:info toc:nil
```

The HTML header now has the code needed to automatically invoke the script. For setting options, use the syntax from the above line for options described below:

'path:'
The path to the script. The default is to grab the script from https://orgmode.org/org-info.js, but you might want to have a local copy and use a path like '`../scripts/org-info.js`'.

'view:'
Initial view when the website is first shown. Possible values are:

'info'	Info-like interface with one section per page
'overview'	Folding interface, initially showing only top-level
'content'	Folding interface, starting with all headlines visible
'showall'	Folding interface, all headlines and text visible

'sdepth:'

Maximum headline level still considered as an independent section for info and folding modes. The default is taken from `org-export-headline-levels`, i.e., the 'H' switch in 'OPTIONS'. If this is smaller than in `org-export-headline-levels`, each info/folding section can still contain child headlines.

'toc:'
Should the table of contents *initially* be visible? Even when 'nil', you can always get to the "toc" with `i`.

'tdepth:'
The depth of the table of contents. The defaults are taken from the variables `org-export-headline-levels` and `org-export-with-toc`.

'ftoc:'
Does the CSS of the page specify a fixed position for the "toc"? If yes, the toc is displayed as a section.

'ltoc:'
Should there be short contents (children) in each section? Make this 'above' if the section should be above initial text.

'mouse:'
Headings are highlighted when the mouse is over them. Should be 'underline' (default) or a background color like '#cccccc'.

'buttons:'
Should view-toggle buttons be everywhere? When 'nil' (the default), only one such button is present.

You can choose default values for these options by customizing the variable `org-infojs-options`. If you always want to apply the script to your pages, configure the variable `org-export-html-use-infojs`.

Next: Markdown Export, Previous: HTML Export, Up: Exporting [Contents][Index]

13.10 LaTeX Export

The LaTeX export backend can handle complex documents, incorporate standard or custom LaTeX document classes, generate documents using alternate LaTeX engines, and produce fully linked PDF files with indexes, bibliographies, and tables of contents, destined for interactive online viewing or high-quality print publication.

While the details are covered in-depth in this section, here are some quick references to variables for the impatient: for engines, see `org-latex-compiler`; for build sequences, see `org-latex-pdf-process`; for packages, see `org-latex-default-packages-alist` and `org-latex-packages-alist`.

An important note about the LaTeX export backend: it is sensitive to blank lines in the Org document. That's because LaTeX itself depends on blank lines to tell apart syntactical elements, such as paragraphs.

The following sections expect users to be familiar with common LaTeX terminology. You may refer to https://tug.org/begin.html to get familiar with LaTeX basics. Users with LaTeX installed may also run '`texdoc latex`' from terminal to open LaTeX introduction [137]

- LaTeX/PDF export commands
- LaTeX specific export settings
- LaTeX header and sectioning structure
- Quoting LaTeX code
- Tables in LaTeX export
- Images in LaTeX export
- Plain lists in LaTeX export
- Source blocks in LaTeX export
- Example blocks in LaTeX export
- Special blocks in LaTeX export
- Horizontal rules in LaTeX export
- Verse blocks in LaTeX export
- Quote blocks in LaTeX export

13.10.1 LaTeX/PDF export commands

`C-c C-e l l` (`org-latex-export-to-latex`) ¶
Export to a LaTeX file with a '`.tex`' extension. For '`myfile.org`', Org exports to '`myfile.tex`', overwriting without warning.

`C-c C-e l L` (`org-latex-export-as-latex`) ¶
Export to a temporary buffer. Do not create a file.

`C-c C-e l p` (`org-latex-export-to-pdf`) ¶
Export as LaTeX file and convert it to PDF file.

`C-c C-e l o` ¶
Export as LaTeX file and convert it to PDF, then open the PDF using the default viewer.

`M-x org-export-region-as-latex`
Convert the region to LaTeX under the assumption that it was in Org mode syntax before. This is a global command that can be invoked in any buffer.

The LaTeX export backend can use any of these LaTeX engines: '`pdflatex`', '`xelatex`', and '`lualatex`'. These engines compile LaTeX files with different compilers, packages, and output options. The LaTeX export backend finds the compiler version to use from `org-latex-compiler` variable or the '`#+LATEX_COMPILER`' keyword in the Org file. See the docstring for the `org-latex-default-packages-alist` for loading packages with certain compilers. Also see `org-latex-bibtex-compiler` to set the bibliography compiler[138].

13.10.2 LaTeX specific export settings

The LaTeX export backend has several additional keywords for customizing LaTeX output. Setting these keywords works similar to the general options (see Export Settings).

'DESCRIPTION' ¶

The document's description. The description along with author name, keywords, and related file metadata are inserted in the output file by the hyperref package. See `org-latex-hyperref-template` for customizing metadata items. See `org-latex-title-command` for typesetting description into the document's front matter. Use multiple '`DESCRIPTION`' keywords for long descriptions.

'LANGUAGE' ¶

Language code of the primary document language. When '`LANGUAGE`' keyword is not not specified use the value of `org-export-default-language` (by default - '`en`', American English)

The list of language codes supported by Org is stored in the variable `org-latex-language-alist`.

In order to be effective, the '`babel`' or '`polyglossia`' packages—according to the LaTeX compiler used—must be loaded with the appropriate language as argument. This can be accomplished by modifying the `org-latex-packages-alist` variable, e.g., with the following snippet (note that '`polyglossia`' does not work with pdfLaTeX):

```
(add-to-list 'org-latex-packages-alist
             '("AUTO" "babel" t ("pdflatex" "xelatex" "lualatex")))
(add-to-list 'org-latex-packages-alist
             '("AUTO" "polyglossia" t ("xelatex" "lualatex")))
```

'LATEX_CLASS' ¶

This is LaTeX document class, such as *article*, *report*, *book*, and so on, which contain predefined preamble and headline level mapping that the LaTeX export backend needs. The backend reads the default class name from the `org-latex-default-class` variable. Org has *article* as the default class. A valid default class must be an element of `org-latex-classes`.

'LATEX_CLASS_OPTIONS' ¶

Options the LaTeX export backend uses when calling the LaTeX document class.

'LATEX_COMPILER' ¶

The compiler, such as '`pdflatex`', '`xelatex`', '`lualatex`', for producing the PDF. See `org-latex-compiler`.

'LATEX_HEADER_EXTRA'

Arbitrary lines to add to the document's preamble, before the hyperref settings. See `org-latex-classes` for adjusting the structure and order of the LaTeX headers.

'KEYWORDS' ¶

The keywords for the document. The description along with author name, keywords, and related file metadata are inserted in the output file by the hyperref package. See `org-latex-hyperref-template` for customizing metadata items. See `org-latex-title-command` for typesetting description into the document's front matter. Use multiple '`KEYWORDS`' lines if necessary.

'SUBTITLE' ¶

The document's subtitle. It is typeset as per `org-latex-subtitle-format`. If `org-latex-subtitle-separate` is non-`nil`, it is typed outside of the `\title` macro. See `org-latex-hyperref-template` for customizing metadata items. See `org-latex-title-command` for

typesetting description into the document's front matter.

The following sections have further details.

Next: Quoting LaTeX code, Previous: LaTeX specific export settings, Up: LaTeX Export [Contents][Index]

13.10.3 LaTeX header and sectioning structure

The LaTeX export backend converts the first three of Org's outline levels into LaTeX headlines. The remaining Org levels are exported as lists. To change this globally for the cut-off point between levels and lists, (see Export Settings).

By default, the LaTeX export backend uses the *article* class.

To change the default class globally, edit `org-latex-default-class`. To change the default class locally in an Org file, add option lines '`#+LATEX_CLASS: myclass`'. To change the default class for just a part of the Org file, set a subtree property, 'EXPORT_LATEX_CLASS'. The class name entered here must be valid member of `org-latex-classes`. This variable defines a header template for each class into which the exporter splices the values of `org-latex-default-packages-alist` and `org-latex-packages-alist`. Use the same three variables to define custom sectioning or custom classes.

The LaTeX export backend sends the 'LATEX_CLASS_OPTIONS' keyword and 'EXPORT_LATEX_CLASS_OPTIONS' property as options to the LaTeX `\documentclass` macro. The options and the syntax for specifying them, including enclosing them in square brackets, follow LaTeX conventions.

```
#+LATEX_CLASS_OPTIONS: [a4paper,11pt,twoside,twocolumn]
```

The LaTeX export backend appends values from 'LATEX_HEADER' and 'LATEX_HEADER_EXTRA' keywords to the LaTeX header. The docstring for `org-latex-classes` explains in more detail. Also note that LaTeX export backend does not append 'LATEX_HEADER_EXTRA' to the header when previewing LaTeX snippets (see Previewing LaTeX fragments).

A sample Org file with the above headers:

```
#+LATEX_CLASS: article
#+LATEX_CLASS_OPTIONS: [a4paper]
#+LATEX_HEADER: \usepackage{xyz}

* Headline 1
  some text
* Headline 2
  some more text
```

LaTeX packages '`babel`' or '`polyglossia`' can also be loaded in a document. The "AUTO" string will be replaced in both cases by the appropriate value for the 'LANGUAGE' keyword, if present in the document, or by the value of `org-export-default-language`. Let's see some examples in one or another case.

‘Babel’ accepts the classic syntax and (in addition) the new syntax with the ‘\babelprovide’ command to load the languages using the new ‘INI’ files procedure. Keep in mind that there are a number of languages that are only served in babel using ‘INI’ files, so they cannot be declared using the classic syntax, but only using the ‘\babelprovide’ command (see https://mirrors.ctan.org/macros/latex/required/babel/base/babel.pdf). Valid usage examples could be:

```
#+LATEX_HEADER: \usepackage[french,italian,AUTO]{babel}
```

where “AUTO” is the main language. But it can also be loaded using the ‘\babelprovide’ command:

```
#+LATEX_HEADER: \usepackage[french,italian]{babel}
#+LATEX_HEADER: \babelprovide[import, main]{AUTO}
```

‘Polyglossia’, for this procedure to be effective, must be loaded using the same ‘babel’ classic syntax (but note that *this is not* the actual polyglossia syntax). For example, suppose a document declares Polytonic Greek as the primary language, and French as the secondary language. In this case, it would be expressed as:

```
#+LANGUAGE: el-polyton
#+LATEX_HEADER: \usepackage[french,AUTO]{polyglossia}
```

This would produce in LaTeX (with the actual ‘polyglossia’ syntax):

```
\usepackage{polyglossia}
\setmainlanguage[variant=polytonic]{greek}
\setotherlanguage{french}
```

13.10.4 Quoting LaTeX code

When the available LaTeX export customizations are not sufficient to fine-tune the desired output, it is possible to insert any arbitrary LaTeX code (see Embedded LaTeX). There are three ways to embed such code in the Org file and they all use different quoting syntax.

Inserting in-line quoted with @ symbols:

```
Code embedded in-line @@latex:any arbitrary LaTeX code@@ in a paragraph.
```

Inserting as one or more keyword lines in the Org file:

```
#+LATEX: any arbitrary LaTeX code
```

Inserting as an export block in the Org file, where the backend exports any code between begin and end markers:

```
#+BEGIN_EXPORT latex
  any arbitrary LaTeX code
#+END_EXPORT
```

13.10.5 Tables in LaTeX export

The LaTeX export backend can pass several LaTeX attributes for table contents and layout. Besides specifying a label (see Internal Links) and a caption (see Captions), the other valid LaTeX attributes include:

‘:mode’ ¶
The LaTeX export backend wraps the table differently depending on the mode for accurate rendering of math symbols. Mode is either ‘`table`’, ‘`math`’, ‘`inline-math`’, ‘`verbatim`’ or ‘`tabbing`’.

For ‘`math`’ or ‘`inline-math`’ mode, LaTeX export backend wraps the table in a math environment, but every cell in it is exported as-is. For ‘`tabbing`’ the LaTeX tabbing environment is used and the correct tabbing delimiters ‘`\>`’ are used. The LaTeX export backend determines the default mode from `org-latex-default-table-mode`. The LaTeX export backend merges contiguous tables in the same mode into a single environment.

‘:environment’ ¶
Set the default LaTeX table environment for the LaTeX export backend to use when exporting Org tables. Common LaTeX table environments are provided by these packages: tabularx, longtable, array, tabu, and bmatrix. For packages, such as tabularx and tabu, or any newer replacements, include them in the `org-latex-packages-alist` variable so the LaTeX export backend can insert the appropriate load package headers in the converted LaTeX file. Look in the docstring for the `org-latex-packages-alist` variable for configuring these packages for LaTeX snippet previews, if any.

‘:caption’
Use ‘`CAPTION`’ keyword to set a simple caption for a table (see Captions). For custom captions, use ‘`:caption`’ attribute, which accepts raw LaTeX code. ‘`:caption`’ value overrides ‘`CAPTION`’ value.

‘:float’
‘:placement’
The table environments by default are not floats in LaTeX. To make them floating objects use ‘`:float`’ with one of the following options: ‘`t`’ (for a default ‘`table`’ environment), ‘`sideways`’ (for a ‘`sidewaystable`’ environment), ‘`multicolumn`’ (to span the table across multiple columns of a page in a ‘`table*`’ environment) and ‘`nil`’. In addition to these three values, ‘`:float`’ can pass through any arbitrary value, for example a user-defined float type with the ‘`float`’ LaTeX package.

LaTeX floats can also have additional layout ‘`:placement`’ attributes. These are the usual ‘`[h t b p ! H]`’ permissions specified in square brackets. Note that for ‘`:float sideways`’ tables, the LaTeX export backend ignores ‘`:placement`’ attributes.

‘:align’
‘:font’
‘:width’
The LaTeX export backend uses these attributes for regular tables to set their alignments, fonts, and widths.

‘:options’
The ‘`:options`’ attribute allows adding an optional argument with a list of various table options (between brackets in LaTeX export), since certain tabular environments, such as longtblr of the tabularray LaTeX package, provides this structure. For example: ‘`:options remark{Note}=`

`{note},remark{Source}={source}`'.

':spread'
When ':spread' is non-nil, the LaTeX export backend spreads or shrinks the table by the ':width' for tabu and longtabu environments. ':spread' has no effect if ':width' is not set.

':booktabs' ¶
':center'
':rmlines'
All three commands are toggles. ':booktabs' brings in modern typesetting enhancements to regular tables. The booktabs package has to be loaded through `org-latex-packages-alist`. ':center' is for centering the table. ':rmlines' removes all but the very first horizontal line made of ASCII characters from "table.el" tables only.

':math-prefix'
':math-suffix'
':math-arguments'
The LaTeX export backend inserts ':math-prefix' string value in a math environment before the table. The LaTeX export backend inserts ':math-suffix' string value in a math environment after the table. The LaTeX export backend inserts ':math-arguments' string value between the macro name and the table's contents. ':math-arguments' comes in use for matrix macros that require more than one argument, such as 'qbordermatrix'.

LaTeX table attributes help formatting tables for a wide range of situations, such as matrix product or spanning multiple pages:

```
#+ATTR_LATEX: :environment longtable :align l|lp{3cm}r|l
| ... | ... |
| ... | ... |

#+ATTR_LATEX: :mode math :environment bmatrix :math-suffix \times
| a | b |
| c | d |
#+ATTR_LATEX: :mode math :environment bmatrix
| 1 | 2 |
| 3 | 4 |
```

Set the caption with the LaTeX command '`\bicaption{HeadingA}{HeadingB}`':

```
#+ATTR_LATEX: :caption \bicaption{HeadingA}{HeadingB}
| ... | ... |
| ... | ... |
```

13.10.6 Images in LaTeX export

The LaTeX export backend processes image links in Org files that do not have descriptions, such as these links '`[[file:img.jpg]]`' or '`[[./img.jpg]]`', as direct image insertions in the final PDF output. In the PDF, they are no longer links but actual images embedded on the page. The LaTeX export backend uses '`\includegraphics`' macro to insert the image. But for TikZ (https://sourceforge.net/projects/pgf/) images, the backend uses an `\input` macro wrapped within a `tikzpicture` environment.

For specifying image ‘`:width`’, ‘`:height`’, ‘`:scale`’ and other ‘`:options`’, use this syntax:

```
#+ATTR_LATEX: :width 5cm :options angle=90
[[./img/sed-hr4049.pdf]]
```

A ‘`:scale`’ attribute overrides both ‘`:width`’ and ‘`:height`’ attributes.

For custom commands for captions, use the ‘`:caption`’ attribute. It overrides the default ‘`#+CAPTION`’ value:

```
#+ATTR_LATEX: :caption \bicaption{HeadingA}{HeadingB}
[[./img/sed-hr4049.pdf]]
```

When captions follow the method as described in Captions, the LaTeX export backend wraps the picture in a floating ‘`figure`’ environment. To float an image without specifying a caption, set the ‘`:float`’ attribute to one of the following:

‘`t`’
For a default ‘`figure`’ environment.

‘`multicolumn`’
To span the image across multiple columns of a page; the backend wraps the image in a ‘`figure*`’ environment.

‘`wrap`’
For text to flow around the image on the right; the figure occupies the left half of the page.

‘`sideways`’
For a new page with the image sideways, rotated ninety degrees, in a ‘`sidewaysfigure`’ environment; overrides ‘`:placement`’ setting.

‘`nil`’
To avoid a ‘`:float`’ even if using a caption.

Any arbitrary value
For example, a user-defined float type with the ‘`float`’ LaTeX package.

Use the ‘`placement`’ attribute to modify a floating environment’s placement.

```
#+ATTR_LATEX: :float wrap :width 0.38\textwidth :placement {r}{0.4\textwidth}
[[./img/hst.png]]
```

The LaTeX export backend centers all images by default. Setting ‘`:center`’ to ‘`nil`’ disables centering. To disable centering globally, set `org-latex-images-centered` to ‘`nil`’.

Set the ‘`:comment-include`’ attribute to non-`nil` value for the LaTeX export backend to comment out the ‘`\includegraphics`’ macro.

13.10.7 Plain lists in LaTeX export

The LaTeX export backend accepts the ‘`environment`’ and ‘`options`’ attributes for plain lists. Both attributes work together for customizing lists, as shown in the examples:

```
#+LATEX_HEADER: \usepackage[inline]{enumitem}
Some ways to say "Hello":
#+ATTR_LATEX: :environment itemize*
#+ATTR_LATEX: :options [label={}, itemjoin={,}, itemjoin*={, and}]
- Hola
- Bonjour
- Guten Tag.
```

Since LaTeX supports only four levels of nesting for lists, use an external package, such as ‘`enumitem`’ in LaTeX, for levels deeper than four:

```
#+LATEX_HEADER: \usepackage{enumitem}
#+LATEX_HEADER: \renewlist{itemize}{itemize}{9}
#+LATEX_HEADER: \setlist[itemize]{label=$\circ$}
- One
  - Two
    - Three
      - Four
        - Five
```

13.10.8 Source blocks in LaTeX export

LaTeX export backend provides multiple ways to render src blocks in LaTeX, according to the value of `org-latex-src-block-backend`. The default value ‘`verbatim`’ renders the src code verbatim, without any extra styling. Alternative values allow more colorful styling, but require additional LaTeX (‘`listings`’, ‘`minted`’), system (‘`minted`’), or Emacs (‘`engraved`’) packages. See the `org-latex-src-block-backend` docstring for more details.

The LaTeX export backend can make source code blocks into floating objects through the attributes ‘`:float`’ and ‘`:options`’. For ‘`:float`’:

‘`t`’
Makes a source block float; by default floats any source block with a caption.

‘`multicolumn`’
Spans the source block across multiple columns of a page.

‘`nil`’
Avoids a ‘`:float`’ even if using a caption; useful for source code blocks that may not fit on a page.

```
#+ATTR_LATEX: :float nil
#+BEGIN_SRC emacs-lisp
  Lisp code that may not fit in a single page.
#+END_SRC
```

The LaTeX export backend passes string values in ‘`:options`’ to LaTeX packages for customization of that specific source block. In the example below, the ‘`:options`’ are set for Engraved or Minted. Minted is a source code highlighting LaTeX package with many configurable options[139]. Both Minted and Engraved are built on fvextra, and so support many of the same options.

```
#+ATTR_LATEX: :options mathescape
#+BEGIN_SRC emacs-lisp
  (defun Fib (n) ; $n_i = n_{i-2} + n_{i-1}$
    (if (< n 2) n (+ (Fib (- n 1)) (Fib (- n 2)))))
#+END_SRC
```

To apply similar configuration options for all source blocks in a file, use the `org-latex-listings-options`, `org-latex-engraved-options`, and `org-latex-minted-options` variables.

13.10.9 Example blocks in LaTeX export

The LaTeX export backend wraps the contents of example blocks in a '`verbatim`' environment. To change this behavior to use another environment globally, specify an appropriate export filter (see Advanced Export Configuration). To change this behavior to use another environment for each block, use the '`:environment`' parameter to specify a custom environment.

```
#+ATTR_LATEX: :environment myverbatim
#+BEGIN_EXAMPLE
  This sentence is false.
#+END_EXAMPLE
```

13.10.10 Special blocks in LaTeX export

For other special blocks in the Org file, the LaTeX export backend makes a special environment of the same name. The backend also takes '`:options`', if any, and appends as-is to that environment's opening string. For example:

```
#+BEGIN_abstract
  We demonstrate how to solve the Syracuse problem.
#+END_abstract

#+ATTR_LATEX: :options [Proof of important theorem]
#+BEGIN_proof
  ...
  Therefore, any even number greater than 2 is the sum of two primes.
#+END_proof
```

exports to

```
\begin{abstract}
  We demonstrate how to solve the Syracuse problem.
\end{abstract}

\begin{proof}[Proof of important theorem]
  ...
  Therefore, any even number greater than 2 is the sum of two primes.
\end{proof}
```

If you need to insert a specific caption command, use '`:caption`' attribute. It overrides standard '`CAPTION`' value, if any. For example:

```
#+ATTR_LATEX: :caption \MyCaption{HeadingA}
#+BEGIN_proof
  ...
#+END_proof
```

13.10.11 Horizontal rules in LaTeX export

The LaTeX export backend converts horizontal rules by the specified ‘`:width`’ and ‘`:thickness`’ attributes. For example:

```
#+ATTR_LATEX: :width .6\textwidth :thickness 0.8pt
-----
```

13.10.12 Verse blocks in LaTeX export

The LaTeX export backend accepts five attributes for verse blocks: ‘`:lines`’, ‘`:center`’, ‘`:versewidth`’, ‘`:latexcode`’ and ‘`:literal`’. The three first require the external LaTeX package ‘`verse.sty`’, which is an extension of the standard LaTeX environment.

‘`:lines`’
To add marginal verse numbering. Its value is an integer, the sequence in which the verses should be numbered.

‘`:center`’
With value ‘`t`’ all the verses on the page are optically centered (a typographic convention for poetry), taking as a reference the longest verse, which must be indicated by the attribute ‘`:versewidth`’.

‘`:versewidth`’
Its value is a literal text string with the longest verse.

‘`:latexcode`’
It accepts any arbitrary LaTeX code that can be included within a LaTeX ‘`verse`’ environment.

‘`:literal`’
With value t, all blank lines are preserved and exported as ‘`\vspace*{\baselineskip}`’, including the blank lines before or after contents. Note that without the ‘`:literal`’ attribute, one or more blank lines between stanzas are exported as a single blank line, and any blank lines before or after the content are removed, which is more consistent with the syntax of the LaTeX ‘verse’ environment, and the one provided by the ‘`verse`’ package. If the ‘`verse`’ package is loaded, the vertical spacing between all stanzas can be controlled by the global length ‘`\stanzaskip`’ (see https://www.ctan.org/pkg/verse).

A complete example with Shakespeare’s first sonnet:

```
#+ATTR_LATEX: :center t :latexcode \color{red} :lines 5
#+ATTR_LATEX: :versewidth Feed’st thy light’s flame with self-substantial fuel,
#+BEGIN_VERSE
From fairest creatures we desire increase,
That thereby beauty’s rose might never die,
But as the riper should by time decease
His tender heir might bear his memory
But thou, contracted to thine own bright eyes,
Feed’st thy light’s flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel.
Thou that art now the world’s fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And, tender churl, mak’st waste in niggardly.
Pity the world, or else this glutton be,
To eat the world’s due, by the grave and thee.
#+END_VERSE
```

13.10.13 Quote blocks in LaTeX export

The LaTeX export backend accepts two attributes for quote blocks: ‘`:environment`’, for an arbitrary quoting environment (the default value is that of `org-latex-default-quote-environment`: `"quote"`) and ‘`:options`’. For example, to choose the environment ‘`quotation`’, included as an alternative to ‘`quote`’ in standard LaTeX classes:

```
#+ATTR_LATEX: :environment quotation
#+BEGIN_QUOTE
some text...
#+END_QUOTE
```

To choose the ‘`foreigndisplayquote`’ environment, included in the LaTeX package ‘`csquotes`’, with the ‘`german`’ option, use this syntax:

```
#+LATEX_HEADER:\usepackage[autostyle=true]{csquotes}
#+ATTR_LATEX: :environment foreigndisplayquote :options {german}
#+BEGIN_QUOTE
some text in German...
#+END_QUOTE
```

which is exported to LaTeX as

```
\begin{foreigndisplayquote}{german}
some text in German...
\end{foreigndisplayquote}
```

Next: OpenDocument Text Export, Previous: LaTeX Export, Up: Exporting [Contents][Index]

13.11 Markdown Export

The Markdown export backend, “md”, converts an Org file to Markdown format, as defined at https://daringfireball.net/projects/markdown/. This is the original Markdown specification, developed by John Gruber and Aaron Swartz.

Since “md” backend is built on top of the HTML backend (see HTML Export), it converts every Org construct not defined in Markdown syntax, such as tables, to HTML.

Do note that the original markdown syntax has differences with other commonly used Markdown flavors. See https://en.wikipedia.org/wiki/Markdown for more details.

Markdown export commands

C-c C-e m m (**org-md-export-to-markdown**) ¶
Export to a text file with Markdown syntax. For 'myfile.org', Org exports to 'myfile.md', overwritten without warning.

C-c C-e m M (**org-md-export-as-markdown**) ¶
Export to a temporary buffer. Does not create a file.

C-c C-e m o ¶
Export as a text file with Markdown syntax, then open it.

Header and sectioning structure

Based on `org-md-headline-style`, Markdown export can generate headlines of both *atx* and *setext* types. *setext* limits headline levels to two whereas *atx* limits headline levels to six. *mixed* exports headline levels one and two in *setext*-style, and headline levels three through six as *atx*-style headlines. Beyond these limits, the export backend converts headlines to lists. To set a limit to a level before the absolute limit (see Export Settings).

13.12 OpenDocument Text Export

The ODT export backend handles creating of OpenDocument Text (ODT) format. Documents created by this exporter use the *OpenDocument-v1.2 specification*[140] and are compatible with LibreOffice 3.4.

- Pre-requisites for ODT export
- ODT export commands
- ODT specific export settings
- Extending ODT export
- Applying custom styles
- Links in ODT export
- Tables in ODT export
- Images in ODT export
- Math formatting in ODT export
- Labels and captions in ODT export
- Literal examples in ODT export
- Advanced topics in ODT export

13.12.1 Pre-requisites for ODT export

The ODT export backend relies on the zip program to create the final compressed ODT output. Check if 'zip' is locally available and executable. Without it, export cannot finish.

13.12.2 ODT export commands

C-c C-e o o (**org-odt-export-to-odt**) ¶
Export as OpenDocument Text file.

If `org-odt-preferred-output-format` is specified, the ODT export backend automatically converts the exported file to that format.

For ‘`myfile.org`’, Org exports to ‘`myfile.odt`’, overwriting without warning. The ODT export backend exports a region only if a region was active.

If the selected region is a single tree, the ODT export backend makes the tree head the document title. Incidentally, `C-c @` selects the current subtree. If the tree head entry has, or inherits, an ‘`EXPORT_FILE_NAME`’ property, the ODT export backend uses that for file name.

`C-c C-e o O` ¶
Export as an OpenDocument Text file and open the resulting file.

If `org-export-odt-preferred-output-format` is specified, open the converted file instead. See Automatically exporting to other formats.

13.12.3 ODT specific export settings

The ODT export backend has several additional keywords for customizing ODT output. Setting these keywords works similar to the general options (see Export Settings).

‘`DESCRIPTION`’ ¶
This is the document’s description, which the ODT export backend inserts as document metadata. For long descriptions, use multiple lines, prefixed with ‘`DESCRIPTION`’.

‘`KEYWORDS`’ ¶
The keywords for the document. The ODT export backend inserts the description along with author name, keywords, and related file metadata as metadata in the output file. Use multiple ‘`KEYWORDS`’ if necessary.

‘`ODT_STYLES_FILE`’ ¶
The ODT export backend uses the `org-odt-styles-file` by default. See Applying custom styles for details.

‘`SUBTITLE`’ ¶
The document subtitle.

Next: Applying custom styles, Previous: ODT specific export settings, Up: OpenDocument Text Export [Contents][Index]

13.12.4 Extending ODT export

The ODT export backend can produce documents in other formats besides ODT using a specialized ODT converter process. Its common interface works with popular converters to produce formats such as ‘`doc`’, or convert a document from one format, say ‘`csv`’, to another format, say ‘`xls`’.

Customize `org-odt-convert-process` variable to point to ‘`unoconv`’, which is the ODT’s preferred converter. Working installations of LibreOffice would already have ‘`unoconv`’ installed. Alternatively, other converters may be substituted here. See Configuring a document converter.

Automatically exporting to other formats

If ODT format is just an intermediate step to get to other formats, such as ‘doc’, ‘docx’, ‘rtf’, or ‘pdf’, etc., then extend the ODT export backend to directly produce that format. Specify the final format in the `org-odt-preferred-output-format` variable. This is one way to extend (see ODT export commands).

Converting between document formats

The Org export backend is made to be inter-operable with a wide range of text document format converters. Newer generation converters, such as LibreOffice and Pandoc, can handle hundreds of formats at once. Org provides a consistent interaction with whatever converter is installed. Here are some generic commands:

`M-x org-odt-convert` ¶
Convert an existing document from one format to another. With a prefix argument, opens the newly produced file.

Next: Links in ODT export, Previous: Extending ODT export, Up: OpenDocument Text Export [Contents][Index]

13.12.5 Applying custom styles

The ODT export backend comes with many OpenDocument styles (see Working with OpenDocument style files). To expand or further customize these built-in style sheets, either edit the style sheets directly or generate them using an application such as LibreOffice. The example here shows creating a style using LibreOffice.

Applying custom styles: the easy way

1. Create a sample ‘`example.org`’ file with settings as shown below, and export it to ODT format.

   ```
   #+OPTIONS: H:10 num:t
   ```

2. Open the above ‘`example.odt`’ using LibreOffice. Use the *Stylist* to locate the target styles, which typically have the “Org” prefix. Open one, modify, and save as either OpenDocument Text (ODT) or OpenDocument Template (OTT) file.
3. Customize the variable `org-odt-styles-file` and point it to the newly created file. For additional configuration options, see Overriding factory styles.
   To apply an ODT style to a particular file, use the ‘`ODT_STYLES_FILE`’ keyword as shown in the example below:

   ```
   #+ODT_STYLES_FILE: "/path/to/example.ott"
   ```

   or

   ```
   #+ODT_STYLES_FILE: ("/path/to/file.ott" ("styles.xml" "image/hdr.png"))
   ```

Using third-party styles and templates

The ODT export backend relies on many templates and style names. Using third-party styles and templates can lead to mismatches. Templates derived from built-in ODT templates and styles seem to have fewer problems.

13.12.6 Links in ODT export

ODT exporter creates native cross-references for internal links. It creates Internet-style links for all other links.

A link with no description and pointing to a regular, un-itemized, outline heading is replaced with a cross-reference and section number of the heading.

A '`\ref{label}`'-style reference to an image, table etc., is replaced with a cross-reference and sequence number of the labeled entity. See Labels and captions in ODT export.

13.12.7 Tables in ODT export

The ODT export backend handles native Org mode tables (see Tables) and simple '`table.el`' tables. Complex '`table.el`' tables having column or row spans are not supported. Such tables are stripped from the exported document.

By default, the ODT export backend exports a table with top and bottom frames and with ruled lines separating row and column groups (see Column Groups). All tables are typeset to occupy the same width. The ODT export backend honors any table alignments and relative widths for columns (see Column Width and Alignment).

Note that the ODT export backend interprets column widths as weighted ratios, the default weight being 1.

Specifying '`:rel-width`' property on an '`ATTR_ODT`' line controls the width of the table. For example:

```
#+ATTR_ODT: :rel-width 50
| Area/Month    |   Jan |   Feb |   Mar |   Sum |
|---------------+-------+-------+-------+-------|
| /             |     < |       |       |     < |
| <l13>         |  <r5> |  <r5> |  <r5> |  <r6> |
| North America |     1 |    21 |   926 |   948 |
| Middle East   |     6 |    75 |   844 |   925 |
| Asia Pacific  |     9 |    27 |   790 |   826 |
|---------------+-------+-------+-------+-------|
| Sum           |    16 |   123 |  2560 |  2699 |
```

On export, the above table takes 50% of text width area. The exporter sizes the columns in the ratio: 13:5:5:5:6. The first column is left-aligned and rest of the columns, right-aligned. Vertical rules separate the header and the last column. Horizontal rules separate the header and the last row.

For even more customization, create custom table styles and associate them with a table using the '`ATTR_ODT`' keyword. See Customizing tables in ODT export.

Next: Math formatting in ODT export, Previous: Tables in ODT export, Up: OpenDocument Text Export [Contents][Index]

13.12.8 Images in ODT export

Embedding images

The ODT export backend processes image links in Org files that do not have descriptions, such as these links '`[[file:img.jpg]]`' or '`[[./img.jpg]]`', as direct image insertions in the final output. Either of these examples works:

```
[[file:img.png]]
```

```
[[./img.png]]
```

Embedding clickable images

For clickable images, provide a link whose description is another link to an image file. For example, to embed an image '`org-mode-unicorn.png`' which when clicked jumps to https://orgmode.org website, do the following

```
[[https://orgmode.org][./org-mode-unicorn.png]]
```

Sizing and scaling of embedded images

Control the size and scale of the embedded images with the '`ATTR_ODT`' attribute.

The ODT export backend starts with establishing the size of the image in the final document. The dimensions of this size are measured in centimeters. The backend then queries the image file for its dimensions measured in pixels. For this measurement, the backend relies on ImageMagick's identify program or Emacs `create-image` and `image-size` API. ImageMagick is the preferred choice for large file sizes or frequent batch operations. The backend then converts the pixel dimensions using `org-odt-pixels-per-inch` into the familiar 72 dpi or 96 dpi. The default value for this is in `display-pixels-per-inch`, which can be tweaked for better results based on the capabilities of the output device. Here are some common image scaling operations:

Explicitly size the image

To embed '`img.png`' as a 10 cm x 10 cm image, do the following:

```
#+ATTR_ODT: :width 10 :height 10
[[./img.png]]
```

Scale the image

To embed '`img.png`' at half its size, do the following:

```
#+ATTR_ODT: :scale 0.5
[[./img.png]]
```

Scale the image to a specific width

To embed '`img.png`' with a width of 10 cm while retaining the original height:width ratio, do the following:

```
#+ATTR_ODT: :width 10
[[./img.png]]
```

Scale the image to a specific height

To embed '`img.png`' with a height of 10 cm while retaining the original height:width ratio, do the following:

```
#+ATTR_ODT: :height 10
[[./img.png]]
```

Anchoring of images

The ODT export backend can anchor images to ‘`as-char`’, ‘`paragraph`’, or ‘`page`’. Set the preferred anchor using the ‘`:anchor`’ property of the ‘`ATTR_ODT`’ line.

To create an image that is anchored to a page:

```
#+ATTR_ODT: :anchor page
[[./img.png]]
```

13.12.9 Math formatting in ODT export

The ODT exporter has special support for handling math.

- LaTeX math snippets
- MathML and OpenDocument formula files

13.12.9.1 LaTeX math snippets

LaTeX math snippets (see LaTeX fragments) can be embedded in the ODT document in one of the following ways:

MathML ¶
Add this line to the Org file. This option is activated on a per-file basis.

```
#+OPTIONS: tex:t
```

With this option, LaTeX fragments are first converted into MathML fragments using an external LaTeX-to-MathML converter program. The resulting MathML fragments are then embedded as an OpenDocument Formula in the exported document.

You can specify the LaTeX-to-MathML converter by customizing the variables `org-latex-to-mathml-convert-command` and `org-latex-to-mathml-jar-file`.

If you prefer to use MathToWeb[141] as your converter, you can configure the above variables as shown below.

```
(setq org-latex-to-mathml-convert-command
      "java -jar %j -unicode -force -df %o %I"
      org-latex-to-mathml-jar-file
      "/path/to/mathtoweb.jar")
```

or, to use LaTeXML[142] instead,

```
(setq org-latex-to-mathml-convert-command
      "latexmlmath %i --presentationmathml=%o")
```

To quickly verify the reliability of the LaTeX-to-MathML converter, use the following commands:

`M-x org-export-as-odf`
Convert a LaTeX math snippet to an OpenDocument formula (‘`.odf`’) file.

`M-x org-export-as-odf-and-open`

Convert a LaTeX math snippet to an OpenDocument formula (‘`.odf`’) file and open the formula file with the system-registered application.

PNG images ¶
Add this line to the Org file. This option is activated on a per-file basis.

```
#+OPTIONS: tex:dvipng
```

```
#+OPTIONS: tex:dvisvgm
```

or

```
#+OPTIONS: tex:imagemagick
```

Under this option, LaTeX fragments are processed into PNG or SVG images and the resulting images are embedded in the exported document. This method requires dvipng program, dvisvgm or ImageMagick programs.

13.12.9.2 MathML and OpenDocument formula files

When embedding LaTeX math snippets in ODT documents is not reliable, there is one more option to try. Embed an equation by linking to its MathML (‘`.mml`’) source or its OpenDocument formula (‘`.odf`’) file as shown below:

```
[[./equation.mml]]
```

or

```
[[./equation.odf]]
```

13.12.10 Labels and captions in ODT export

ODT format handles labeling and captioning of objects based on their types. Inline images, tables, LaTeX fragments, and Math formulas are numbered and captioned separately. Each object also gets a unique sequence number based on its order of first appearance in the Org file. Each category has its own sequence. A caption is just a label applied to these objects.

```
#+CAPTION: Bell curve
#+NAME:   fig:SED-HR4049
[[./img/a.png]]
```

When rendered, it may show as follows in the exported document:

```
Figure 2: Bell curve
```

To modify the category component of the caption, customize the option `org-odt-category-map-alist`. For example, to tag embedded images with the string “Illustration” instead of the default string “Figure”, use the following setting:

```
(setq org-odt-category-map-alist
      '(("__Figure__" "Illustration" "value" "Figure" org-odt--enumerable-image-p)))
```

With the above modification, the previous example changes to:

```
Illustration 2: Bell curve
```

13.12.11 Literal examples in ODT export

The ODT export backend supports literal examples (see Literal Examples) with full fontification. Internally, the ODT export backend relies on '`htmlfontify.el`' to generate the style definitions needed for fancy listings. The auto-generated styles get '`OrgSrc`' prefix and inherit colors from the faces used by Emacs Font Lock library for that source language.

For custom fontification styles, customize the `org-odt-create-custom-styles-for-srcblocks` option.

To turn off fontification of literal examples, customize the `org-odt-fontify-srcblocks` option.

Previous: Literal examples in ODT export, Up: OpenDocument Text Export [Contents][Index]

13.12.12 Advanced topics in ODT export

The ODT export backend has extensive features useful for power users and frequent uses of ODT formats.

Configuring a document converter

The ODT export backend works with popular converters with little or no extra configuration. See Extending ODT export. The following is for unsupported converters or tweaking existing defaults.

Register the converter ¶
Add the name of the converter to the `org-odt-convert-processes` variable. Note that it also requires how the converter is invoked on the command line. See the variable's docstring for details.

Configure its capabilities ¶
Specify which formats the converter can handle by customizing the variable `org-odt-convert-capabilities`. Use the entry for the default values in this variable for configuring the new converter. Also see its docstring for details.

Choose the converter ¶
Select the newly added converter as the preferred one by customizing the option `org-odt-convert-process`.

Working with OpenDocument style files

This section explores the internals of the ODT exporter; the means by which it produces styled documents; the use of automatic and custom OpenDocument styles.

The ODT exporter relies on two files for generating its output. These files are bundled with the distribution under the directory pointed to by the variable `org-odt-styles-dir`. The two files are:

'`OrgOdtStyles.xml`'
This file contributes to the '`styles.xml`' file of the final ODT document. This file gets modified for the following purposes:

1. To control outline numbering based on user settings;

2. To add styles generated by ‘`htmlfontify.el`’ for fontification of code blocks.

‘`OrgOdtContentTemplate.xml`’
This file contributes to the ‘`content.xml`’ file of the final ODT document. The contents of the Org outline are inserted between the ‘`<office:text>`’ … ‘`</office:text>`’ elements of this file.

Apart from serving as a template file for the final ‘`content.xml`’, the file serves the following purposes:

1. It contains automatic styles for formatting of tables which are referenced by the exporter;
2. It contains ‘`<text:sequence-decl>`’ … ‘`</text:sequence-decl>`’ elements that control numbering of tables, images, equations, and similar entities.

The following two variables control the location from where the ODT exporter picks up the custom styles and content template files. Customize these variables to override the factory styles used by the exporter.

`org-odt-styles-file`
The ODT export backend uses the file pointed to by this variable, such as ‘`styles.xml`’, for the final output. It can take one of the following values:

‘`FILE.xml`’
Use this file instead of the default ‘`styles.xml`’

‘`FILE.odt`’ or ‘`FILE.ott`’
Use the ‘`styles.xml`’ contained in the specified OpenDocument Text or Template file

‘`FILE.odt`’ or ‘`FILE.ott`’ and a subset of included files
Use the ‘`styles.xml`’ contained in the specified OpenDocument Text or Template file. Additionally extract the specified member files and embed those within the final ODT document.

Use this option if the ‘`styles.xml`’ file references additional files like header and footer images.

`nil`
Use the default ‘`styles.xml`’.

`org-odt-content-template-file`
Use this variable to specify the blank ‘`content.xml`’ used in the final output.

Creating one-off styles

The ODT export backend can read embedded raw OpenDocument XML from the Org file. Such direct formatting is useful for one-off instances.

Embedding ODT tags as part of regular text
Enclose OpenDocument syntax in ‘`@@odt:...@@`’ for inline markup. For example, to highlight a region of text do the following:

```
@@odt:<text:span text:style-name="Highlight">This is highlighted
text</text:span>@@.  But this is regular text.
```

Hint: To see the above example in action, edit the ‘`styles.xml`’ (see Factory styles) and add a custom *Highlight* style as shown below:

```
<style:style style:name="Highlight" style:family="text">
  <style:text-properties fo:background-color="#ff0000"/>
</style:style>
```

Embedding a one-line OpenDocument XML ¶

The ODT export backend can read one-liner options with ‘`#+ODT:`’ in the Org file. For example, to force a page break:

```
#+ODT: <text:p text:style-name="PageBreak"/>
```

Hint: To see the above example in action, edit your ‘`styles.xml`’ (see Factory styles) and add a custom ‘`PageBreak`’ style as shown below.

```
<style:style style:name="PageBreak" style:family="paragraph"
             style:parent-style-name="Text_20_body">
  <style:paragraph-properties fo:break-before="page"/>
</style:style>
```

Embedding a block of OpenDocument XML

The ODT export backend can also read ODT export blocks for OpenDocument XML. Such blocks use the ‘`#+BEGIN_EXPORT odt`’ … ‘`#+END_EXPORT`’ constructs.

For example, to create a one-off paragraph that uses bold text, do the following:

```
#+BEGIN_EXPORT odt
  <text:p text:style-name="Text_20_body_20_bold">
  This paragraph is specially formatted and uses bold text.
  </text:p>
#+END_EXPORT
```

Customizing tables in ODT export

Override the default table format by specifying a custom table style with the ‘`#+ATTR_ODT`’ line. For a discussion on default formatting of tables, see Tables in ODT export.

This feature closely mimics the way table templates are defined in the OpenDocument-v1.2 specification[143].

For quick preview of this feature, install the settings below and export the table that follows:

```
(setq org-export-odt-table-styles
      (append org-export-odt-table-styles
              '(("TableWithHeaderRowAndColumn" "Custom"
                 ((use-first-row-styles . t)
                  (use-first-column-styles . t)))
                ("TableWithFirstRowandLastRow" "Custom"
                 ((use-first-row-styles . t)
                  (use-last-row-styles . t))))))

#+ATTR_ODT: :style TableWithHeaderRowAndColumn
| Name  | Phone | Age |
| Peter |  1234 |  17 |
| Anna  |  4321 |  25 |
```

The example above used ‘`Custom`’ template and installed two table styles ‘`TableWithHeaderRowAndColumn`’ and ‘`TableWithFirstRowandLastRow`’. **Important:** The OpenDocument styles needed for producing the above template were pre-defined. They are available in the section marked ‘`Custom Table Template`’ in ‘`OrgOdtContentTemplate.xml`’ (see Factory styles). For adding new templates, define new styles there.

To use this feature proceed as follows:

1. Create a table template[144].
   A table template is set of ‘`table-cell`’ and ‘`paragraph`’ styles for each of the following table cell categories:

   - Body
   - First column
   - Last column
   - First row
   - Last row
   - Even row
   - Odd row
   - Even column
   - Odd Column

   The names for the above styles must be chosen based on the name of the table template using a well-defined convention.

   The naming convention is better illustrated with an example. For a table template with the name ‘`Custom`’, the needed style names are listed in the following table.

Cell type	Cell style	Paragraph style
Body	‘`CustomTableCell`’	‘`CustomTableParagraph`’
First column	‘`CustomFirstColumnTableCell`’	‘`CustomFirstColumnTableParagraph`’
Last column	‘`CustomLastColumnTableCell`’	‘`CustomLastColumnTableParagraph`’
First row	‘`CustomFirstRowTableCell`’	‘`CustomFirstRowTableParagraph`’
Last row	‘`CustomLastRowTableCell`’	‘`CustomLastRowTableParagraph`’
Even row	‘`CustomEvenRowTableCell`’	‘`CustomEvenRowTableParagraph`’
Odd row	‘`CustomOddRowTableCell`’	‘`CustomOddRowTableParagraph`’
Even column	‘`CustomEvenColumnTableCell`’	‘`CustomEvenColumnTableParagraph`’
Odd column	‘`CustomOddColumnTableCell`’	‘`CustomOddColumnTableParagraph`’

   To create a table template with the name ‘`Custom`’, define the above styles in the ‘`<office:automatic-styles>`’ … ‘`</office:automatic-styles>`’ element of the content template file (see Factory styles).

2. Define a table style[145].
   To define a table style, create an entry for the style in the variable `org-odt-table-styles` and specify the following:

   - the name of the table template created in step (1),
   - the set of cell styles in that template that are to be activated.

   For example, the entry below defines two different table styles 'TableWithHeaderRowAndColumn' and 'TableWithFirstRowandLastRow' based on the same template 'Custom'. The styles achieve their intended effect by selectively activating the individual cell styles in that template.

```
(setq org-export-odt-table-styles
      (append org-export-odt-table-styles
              '(("TableWithHeaderRowAndColumn" "Custom"
                 ((use-first-row-styles . t)
                  (use-first-column-styles . t)))
                ("TableWithFirstRowandLastRow" "Custom"
                 ((use-first-row-styles . t)
                  (use-last-row-styles . t))))))
```

3. Associate a table with the table style.
   To do this, specify the table style created in step (2) as part of the 'ATTR_ODT' line as shown below.

```
#+ATTR_ODT: :style TableWithHeaderRowAndColumn
| Name  | Phone | Age |
| Peter |  1234 |  17 |
| Anna  |  4321 |  25 |
```

Validating OpenDocument XML

Sometimes ODT format files may not open due to '`.odt`' file corruption. To verify if such a file is corrupt, validate it against the OpenDocument Relax NG Compact (RNC) syntax schema. But first the '`.odt`' files have to be decompressed using '`zip`'. Note that '`.odt`' files are ZIP archives: (emacs)File Archives. The contents of ODT files are in XML. For general help with validation—and schema-sensitive editing—of XML files: (nxml-mode)Introduction.

Customize `org-odt-schema-dir` to point to a directory with OpenDocument RNC files and the needed schema-locating rules. The ODT export backend takes care of updating the `rng-schema-locating-files`.

Next: Texinfo Export, Previous: OpenDocument Text Export, Up: Exporting [Contents][Index]

13.13 Org Export

org export backend creates a normalized version of the Org document in current buffer. The exporter evaluates Babel code (see Evaluating Code Blocks) and removes content specific to other backends.

Org export commands

`C-c C-e O o` (`org-org-export-to-org`)¶

Export as an Org file with a '`.org`' extension. For '`myfile.org`', Org exports to '`myfile.org.org`', overwriting without warning.

`C-c C-e O v` **(~~)** ¶
Export to an Org file, then open it.

Next: iCalendar Export, Previous: Org Export, Up: Exporting [Contents][Index]

13.14 Texinfo Export

- Texinfo export commands
- Texinfo specific export settings
- Texinfo file header
- Texinfo title and copyright page
- Info directory file
- Headings and sectioning structure
- Indices
- Quoting Texinfo code
- Plain lists in Texinfo export
- Tables in Texinfo export
- Images in Texinfo export
- Quotations in Texinfo export
- Key bindings in Texinfo export
- Special blocks in Texinfo export
- A Texinfo example

13.14.1 Texinfo export commands

`C-c C-e i t` **(`org-texinfo-export-to-texinfo`)** ¶
Export as a Texinfo file with '`.texi`' extension. For '`myfile.org`', Org exports to '`myfile.texi`', overwriting without warning.

`C-c C-e i i` **(`org-texinfo-export-to-info`)** ¶
Export to Texinfo format first and then process it to make an Info file. To generate other formats, such as DocBook, customize the `org-texinfo-info-process` variable.

13.14.2 Texinfo specific export settings

The Texinfo export backend has several additional keywords for customizing Texinfo output. Setting these keywords works similar to the general options (see Export Settings).

'SUBTITLE' ¶
The document subtitle.

'SUBAUTHOR' ¶
Additional authors for the document.

'TEXINFO_FILENAME' ¶
The Texinfo filename.

'TEXINFO_CLASS' ¶

The default document class (`org-texinfo-default-class`), which must be a member of `org-texinfo-classes`.

‘TEXINFO_HEADER’ ¶
Arbitrary lines inserted at the end of the header.

‘TEXINFO_POST_HEADER’ ¶
Arbitrary lines inserted after the end of the header.

‘TEXINFO_DIR_CATEGORY’ ¶
The directory category of the document. Defaults to `Misc`.

‘TEXINFO_DIR_NAME’ ¶
The directory name of the document. This is the short name under which the `m` command will find your manual in the main Info directory. It defaults to the base name of the Texinfo file.

The full form of the Texinfo entry is `* DIRNAME: NODE.` where `NODE` is usually just `(FILENAME)`. Normally this option only provides the `DIRNAME` part, but if you need more control, it can also be the full entry (recognized by the presence of parentheses or a leading ~* ~).

‘TEXINFO_DIR_DESC’ ¶
The directory description of the document. Defaults to the title of the document.

‘TEXINFO_PRINTED_TITLE’ ¶
The printed title of the document.

Next: Texinfo title and copyright page, Previous: Texinfo specific export settings, Up: Texinfo Export [Contents][Index]

13.14.3 Texinfo file header

After creating the header for a Texinfo file, the Texinfo backend automatically generates a name and destination path for the Info file. To override this default with a more sensible path and name, specify the ‘TEXINFO_FILENAME’ keyword.

Along with the output’s file name, the Texinfo header also contains language details (see Export Settings) and encoding system as set in the `org-texinfo-coding-system` variable. Insert ‘TEXINFO_HEADER’ keywords for each additional command in the header, for example:

```
#+TEXINFO_HEADER: @synindex
```

Instead of repeatedly installing the same set of commands, define a class in `org-texinfo-classes` once, and then activate it in the document by setting the ‘TEXINFO_CLASS’ keyword to that class.

Next: Info directory file, Previous: Texinfo file header, Up: Texinfo Export [Contents][Index]

13.14.4 Texinfo title and copyright page

The default template for hard copy output has a title page with ‘TITLE’ and ‘AUTHOR’ keywords (see Export Settings). To replace the regular title with something different for the printed version, use the ‘TEXINFO_PRINTED_TITLE’ and ‘SUBTITLE’ keywords. Both expect raw Texinfo code

for setting their values.

If one ‘AUTHOR’ line is not sufficient, add multiple ‘SUBAUTHOR’ keywords. They have to be set in raw Texinfo code.

```
#+AUTHOR: Jane Smith
#+SUBAUTHOR: John Doe
#+TEXINFO_PRINTED_TITLE: This Long Title@@inlinefmt{tex,@*} Is Broken in @TeX{}
```

Copying material is defined in a dedicated headline with a non-`nil` ‘COPYING’ property. The backend inserts the contents within a ‘`@copying`’ command at the beginning of the document. The heading itself does not appear in the structure of the document.

Copyright information is printed on the back of the title page.

```
* Legalese
  :PROPERTIES:
  :COPYING: t
  :END:

  This is a short example of a complete Texinfo file, version 1.0.

  Copyright \copy 2016 Free Software Foundation, Inc.
```

13.14.5 Info directory file

The end result of the Texinfo export process is the creation of an Info file. This Info file’s metadata has variables for category, title, and description: ‘TEXINFO_DIR_CATEGORY’, ‘TEXINFO_DIR_NAME’, and ‘TEXINFO_DIR_DESC’ keywords that establish where in the Info hierarchy the file fits.

Here is an example that writes to the Info directory file:

```
#+TEXINFO_DIR_CATEGORY: Emacs
#+TEXINFO_DIR_NAME: Org Mode
#+TEXINFO_DIR_DESC: Outline-based notes management and organizer
```

Next: Indices, Previous: Info directory file, Up: Texinfo Export [Contents][Index]

13.14.6 Headings and sectioning structure

The Texinfo export backend uses a pre-defined scheme to convert Org headlines to equivalent Texinfo structuring commands. A scheme like this maps top-level headlines to numbered chapters tagged as `@chapter` and lower-level headlines to unnumbered chapters tagged as `@unnumbered`. To override such mappings to introduce `@part` or other Texinfo structuring commands, define a new class in `org-texinfo-classes`. Activate the new class with the ‘TEXINFO_CLASS’ keyword. When no new class is defined and activated, the Texinfo export backend defaults to the `org-texinfo-default-class`.

If an Org headline’s level has no associated Texinfo structuring command, or is below a certain threshold (see Export Settings), then the Texinfo export backend makes it into a list item.

The Texinfo export backend makes any headline with a non-`nil` ‘`APPENDIX`’ property into an appendix. This happens independent of the Org headline level or the ‘`TEXINFO_CLASS`’ keyword.

The Texinfo export backend creates a menu entry after the Org headline for each regular sectioning structure. To override this with a shorter menu entry, use the ‘`ALT_TITLE`’ property (see Table of Contents). Texinfo menu entries also have an option for a longer ‘`DESCRIPTION`’ property. Here’s an example that uses both to override the default menu entry:

```
* Controlling Screen Display
  :PROPERTIES:
  :ALT_TITLE: Display
  :DESCRIPTION: Controlling Screen Display
  :END:
```

The text before the first headline belongs to the *Top* node, i.e., the node in which a reader enters an Info manual. As such, it is expected not to appear in printed output generated from the ‘`.texi`’ file. See (texinfo)The Top Node, for more information.

13.14.7 Indices

The Texinfo export backend recognizes these indexing keywords if used in the Org file: ‘`CINDEX`’, ‘`FINDEX`’, ‘`KINDEX`’, ‘`PINDEX`’, ‘`TINDEX`’ and ‘`VINDEX`’. Write their value as verbatim Texinfo code; in particular, ‘`{`’, ‘`}`’ and ‘`@`’ characters need to be escaped with ‘`@`’ if they do not belong to a Texinfo command.

```
#+CINDEX: Defining indexing entries
```

For the backend to generate an index entry for a headline, set the ‘`INDEX`’ property to ‘`cp`’ or ‘`vr`’. These abbreviations come from Texinfo that stand for concept index and variable index. The Texinfo manual has abbreviations for all other kinds of indexes. The backend exports the headline as an unnumbered chapter or section command, and then inserts the index after its contents.

```
* Concept Index
  :PROPERTIES:
  :INDEX: cp
  :END:
```

13.14.8 Quoting Texinfo code

Use any of the following three methods to insert or escape raw Texinfo code:

```
Richard @@texinfo:@sc{@@Stallman@@texinfo:}@@ commence' GNU.

#+TEXINFO: @need800
This paragraph is preceded by...

#+BEGIN_EXPORT texinfo
  @auindex Johnson, Mark
  @auindex Lakoff, George
#+END_EXPORT
```

13.14.9 Plain lists in Texinfo export

The Texinfo export backend converts unordered and ordered lists in the Org file using the default command ‘`@itemize`’.

Ordered lists are numbered when exported to Texinfo format. Such numbering obeys any counter (see Plain Lists) in the first item of the list. The ‘`:enum`’ attribute also let you start the list at a specific number, or switch to a lettered list, as illustrated here:

```
#+ATTR_TEXINFO: :enum A
1. Alpha
2. Bravo
3. Charlie
```

The Texinfo export backend by default converts description lists in the Org file using the default command ‘`@table`’, which results in a table with two columns. To change this behavior, set ‘`:table-type`’ attribute to either ‘`ftable`’ or ‘`vtable`’ value. For more information, see (texinfo)Two-column Tables.

The Texinfo export backend by default also applies a text highlight based on the defaults stored in `org-texinfo-table-default-markup`. To override the default highlight command, specify another one with the ‘`:indic`’ attribute.

Org syntax is limited to one entry per list item. Nevertheless, the Texinfo export backend can split that entry according to any text provided through the ‘`:sep`’ attribute. Each part then becomes a new entry in the first column of the table.

The following example illustrates all the attributes above:

```
#+ATTR_TEXINFO: :table-type vtable :sep , :indic asis
- foo, bar :: This is the common text for variables foo and bar.
```

becomes

```
@vtable @asis
@item foo
@itemx bar
This is the common text for variables foo and bar.
@end table
```

The ‘`:compact`’ attribute is an alternative to the ‘`:sep`’ attribute, which allows writing each entry on its own line. If this attribute is non-`nil` and an item in a description list has no body but is followed by another item, then the second item is transcoded to ‘`@itemx`’. This example is transcoded to the same output as above.

```
#+ATTR_TEXINFO: :table-type vtable :indic asis :compact t
- foo ::
- bar ::
  This is the common text for variables foo and bar.
```

Support for this compact syntax can also be enabled for all lists in a file using the ‘`compact-itemx`’ export option, or globally using the variable `org-texinfo-compact-itemx`.

The Texinfo export backend also supports two approaches to writing Texinfo definition commands (see (texinfo)Definition Commands). One of them uses description lists and is described below, the other relies on special blocks (see Special blocks in Texinfo export).

Items in a description list in an Org file that begin with '`Function:`' or certain other prefixes are converted using Texinfo definition commands. This works even if other items in the same list do not have such a prefix; if necessary a single description list is converted using multiple tables (such as '`@vtable`') and definition commands (such as '`@defun`').

```
- Function: org-texinfo-drawer drawer contents info ::
  Transcode a DRAWER element from Org to Texinfo.
```

becomes

```
@defun org-texinfo-drawer drawer contents info ::
  Transcode a DRAWER element from Org to Texinfo.
@end defun
```

The recognized prefixes are '`Command:`', '`Function:`', '`Macro:`', '`Special Form:`', '`Variable:`' and '`User Option:`'. These are the same prefixes that appear in the Info file for the respective definition commands. For example a '`Function:`' item in the Org file is converted to a '`@defun`' command in the Texinfo file, which in turn is converted to a definition prefixed with '`-- Function:`' in the Info file.

As a special case the prefix '`Key:`' is also recognized. No Texinfo definition command exists for key bindings and the output in Info files also lacks the '`Key:`' prefix. Even so this special case is supported because it provides a convenient shorthand, as illustrated here:

```
- Key: C-c C-c (do-something) ::
  This command does something.

- User Option: do-something-somehow ::
  This option controls how exactly ~do-something~ does its thing.
```

becomes

```
@table @asis
@item @kbd{C-c C-c} (@code{do-something})
@kindex C-c C-c
@findex do-something
This command does something.
@end table

@defopt do-something-somehow
This option controls how exactly @code{do-something} does its thing.
@end defopt
```

Command in parenthesis, as done above, is optional.

13.14.10 Tables in Texinfo export

When exporting tables, the Texinfo export backend uses the widest cell width in each column. To override this and instead specify as fractions of line length, use the '`:columns`' attribute. See example below.

```
#+ATTR_TEXINFO: :columns .5 .5
| a cell | another cell |
```

13.14.11 Images in Texinfo export

Insert a file link to the image in the Org file, and the Texinfo export backend inserts the image. These links must have the usual supported image extensions and no descriptions. To scale the image, use ‘`:width`’ and ‘`:height`’ attributes. For alternate text, use ‘`:alt`’ and specify the text using Texinfo code, as shown in the example:

```
#+ATTR_TEXINFO: :width 1in :alt Alternate @i{text}
[[ridt.pdf]]
```

13.14.12 Quotations in Texinfo export

You can write the text of a quotation within a quote block (see Paragraphs). You may also emphasize some text at the beginning of the quotation with the ‘`:tag`’ attribute.

```
#+ATTR_TEXINFO: :tag Warning
#+BEGIN_QUOTE
Striking your thumb with a hammer may cause severe pain and discomfort.
#+END_QUOTE
```

To specify the author of the quotation, use the ‘`:author`’ attribute.

```
#+ATTR_TEXINFO: :author King Arthur
#+BEGIN_QUOTE
The Lady of the Lake, her arm clad in the purest shimmering samite,
held aloft Excalibur from the bosom of the water, signifying by divine
providence that I, Arthur, was to carry Excalibur.  That is why I am
your king.
#+END_QUOTE
```

13.14.13 Key bindings in Texinfo export

Org does not provide any markup for key bindings that corresponds to Texinfo’s `@kbd` and `@key` commands. One way to deal with this is to fall back to code syntax. ‘`~C-x SPC~`’, for example, is transcoded to `@code{C-x SPC}`.

A better approach is to define and use an Org macro named `kbd`. To make that easier the function `org-texinfo-kbd-macro` is provided, which is intended to be used like this:

```
#+macro: kbd (eval (org-texinfo-kbd-macro $1))

Type {{{kbd(C-c SPC)}}}.
```

which becomes

```
Type @kbd{C-c @key{SPC}}.
```

13.14.14 Special blocks in Texinfo export

The Texinfo export backend supports two approaches to writing Texinfo definition commands. One of them is described here, the other in Plain lists in Texinfo export.

The Texinfo export backend converts special blocks to commands with the same name. It also adds any ‘`:options`’ attributes to the end of the command, as shown in this example:

```
#+ATTR_TEXINFO: :options org-org-export-to-org ...
#+BEGIN_defun
  A somewhat obsessive function name.
#+END_defun
```

becomes

```
@defun org-org-export-to-org ...
  A somewhat obsessive function name.
@end defun
```

13.14.15 A Texinfo example

Here is a more detailed example Org file. See (texinfo)GNU Sample Texts for an equivalent example using Texinfo code.

```
#+TITLE: GNU Sample {{{version}}}
#+SUBTITLE: for version {{{version}}}, {{{updated}}}
#+AUTHOR: A.U. Thor
#+EMAIL: bug-sample@gnu.org

#+OPTIONS: ':t toc:t author:t email:t
#+LANGUAGE: en

#+MACRO: version 2.0
#+MACRO: updated last updated 4 March 2014

#+TEXINFO_FILENAME: sample.info
#+TEXINFO_HEADER: @syncodeindex pg cp

#+TEXINFO_DIR_CATEGORY: Texinfo documentation system
#+TEXINFO_DIR_NAME: sample
#+TEXINFO_DIR_DESC: Invoking sample

#+TEXINFO_PRINTED_TITLE: GNU Sample

This manual is for GNU Sample (version {{{version}}},
{{{updated}}}).

* Copying
  :PROPERTIES:
  :COPYING:  t
  :END:

  This manual is for GNU Sample (version {{{version}}},
  {{{updated}}}), which is an example in the Texinfo documentation.

  Copyright \copy 2016 Free Software Foundation, Inc.

  #+BEGIN_QUOTE
  Permission is granted to copy, distribute and/or modify this
  document under the terms of the GNU Free Documentation License,
  Version 1.3 or any later version published by the Free Software
  Foundation; with no Invariant Sections, with no Front-Cover Texts,
  and with no Back-Cover Texts.  A copy of the license is included in
  the section entitled "GNU Free Documentation License".
  #+END_QUOTE

* Invoking sample

  #+PINDEX: sample
  #+CINDEX: invoking @command{sample}

  This is a sample manual.  There is no sample program to invoke, but
  if there were, you could see its basic usage and command line
  options here.

* GNU Free Documentation License
  :PROPERTIES:
  :APPENDIX: t
  :END:

  #+INCLUDE: fdl.org

* Index
  :PROPERTIES:
```

```
:INDEX:    cp
:END:
```

13.15 iCalendar Export

A large part of Org mode's interoperability success is its ability to easily export to or import from external applications. The iCalendar export backend takes calendar data from Org files and exports to the standard iCalendar format.

By default, iCalendar export only includes headings that contain active timestamps or diary sexps[146].

The iCalendar export backend can also incorporate TODO entries based on the configuration of the `org-icalendar-include-todo` variable. The backend exports plain timestamps as 'VEVENT', TODO items as 'VTODO', and also create events from deadlines that are in non-TODO items.

The backend uses the deadlines and scheduling dates in Org TODO items for setting the start and due dates for the iCalendar TODO entry. Consult the `org-icalendar-use-deadline`, `org-icalendar-use-scheduled`, and `org-icalendar-todo-unscheduled-start` variables for more details.

For tags on the headline, the iCalendar export backend makes them into iCalendar categories. To tweak the inheritance of tags and TODO states, configure the variable `org-icalendar-categories`. To assign clock alarms based on time, configure the `org-icalendar-alarm-time` variable.

The iCalendar format standard requires globally unique identifier—or UID—for each entry. The iCalendar export backend creates UIDs during export. To save a copy of the UID in the Org file set the variable `org-icalendar-store-UID`. The backend looks for the 'ID' property of the entry for reusing the same UID for subsequent exports.

Since a single Org entry can result in multiple iCalendar entries—timestamp, deadline, scheduled item, or TODO item—Org adds prefixes to the UID, depending on which part of the Org entry triggered the creation of the iCalendar entry. Prefixing ensures UIDs remains unique, yet enable synchronization programs trace the connections.

`C-c C-e c f` (**`org-icalendar-export-to-ics`**) ¶
Create iCalendar entries from the current Org buffer and store them in the same directory, using a file extension '`.ics`'.

`C-c C-e c a` (**`org-icalendar-export-agenda-files`**) ¶
Create iCalendar entries from Org files in `org-agenda-files` and store in a separate iCalendar file for each Org file.

`C-c C-e c c` (**`org-icalendar-combine-agenda-files`**) ¶
Create a combined iCalendar file from Org files in `org-agenda-files` and write it to `org-icalendar-combined-agenda-file` file name.

The iCalendar export backend includes ‘SUMMARY’, ‘DESCRIPTION’, ‘LOCATION’, ‘TIMEZONE’ and ‘CLASS’ properties from the Org entries when exporting. To force the backend to inherit the ‘LOCATION’, ‘TIMEZONE’ and ‘CLASS’ properties, configure the `org-use-property-inheritance` variable.

‘SUMMARY’, ‘LOCATION’, and ‘DESCRIPTION’ properties can define multi-line summary, location, or description using ‘<PROPERTY>+’ syntax (see Property Syntax):

```
* Meeting at location with multi-line address
<2024-01-08 Mon 14:20-15:00>
:PROPERTIES:
:LOCATION:  Someplace
:LOCATION+: Some Street 5
:LOCATION+: 12345 Small Town
:END:
```

When Org entries do not have ‘SUMMARY’, ‘DESCRIPTION’, ‘LOCATION’ and ‘CLASS’ properties, the iCalendar export backend derives the summary from the headline, and derives the description from the body of the Org item. The `org-icalendar-include-body` variable limits the maximum number of characters of the content are turned into its description.

The ‘TIMEZONE’ property can be used to specify a per-entry time zone, and is applied to any entry with timestamp information. Time zones should be specified as per the IANA time zone database format, e.g., ‘Asia/Almaty’. Alternately, the property value can be ‘UTC’, to force UTC time for this entry only.

The ‘CLASS’ property can be used to specify a per-entry visibility class or access restrictions, and is applied to any entry with class information. The iCalendar standard defines three visibility classes:

‘PUBLIC’
The entry is publicly visible (this is the default).

‘CONFIDENTIAL’
Only a limited group of clients get access to the event.

‘PRIVATE’
The entry can be retrieved only by its owner.

The server should treat unknown class properties the same as ‘PRIVATE’.

The exported iCalendar file can advise clients how often to check for updates. This duration can be set globally with the `org-icalendar-ttl` variable, or on a per-document basis with the ‘ICAL-TTL’ keyword. This option should be set using the iCalendar notation for time durations; consult the docstring of `org-icalendar-ttl` for more details.

Exporting to iCalendar format depends in large part on the capabilities of the destination application. Some are more lenient than others. Consult the Org mode FAQ for advice on specific applications.

13.16 Other Built-in Backends

Other export backends included with Org are:

‘`ox-man.el`’: Export to a man page.

To activate such backends, either customize `org-export-backends` or load directly with ‘`(require 'ox-man)`’. On successful load, the backend adds new keys in the export dispatcher (see The Export Dispatcher).

Follow the comment section of such files, for example, ‘`ox-man.el`’, for usage and configuration details.

Next: Export Region, Previous: Other Built-in Backends, Up: Exporting [Contents][Index]

13.17 Advanced Export Configuration

Export hooks

The export process executes two hooks before the actual exporting begins. The first hook, `org-export-before-processing-functions`, runs before any expansions of macros, Babel code, and include keywords in the buffer. The second hook, `org-export-before-parsing-functions`, runs before the buffer is parsed.

Functions added to these hooks are called with a single argument: the export backend actually used, as a symbol. You may use them for heavy duty structural modifications of the document. For example, you can remove every headline in the buffer during export like this:

```
(defun my-headline-removal (backend)
  "Remove all headlines in the current buffer.
BACKEND is the export backend being used, as a symbol."
  (org-map-entries
   (lambda ()
     (delete-region (point) (line-beginning-position 2))
     ;; We need to tell `org-map-entries' to not skip over heading at
     ;; point.  Otherwise, it would continue from _next_ heading.  See
     ;; the docstring of `org-map-entries' for details.
     (setq org-map-continue-from (point)))))

(add-hook 'org-export-before-parsing-functions #'my-headline-removal)
```

Filters

Filters are lists of functions to be applied to certain parts for a given backend. The output from the first function in the filter is passed on to the next function in the filter. The final output is the output from the final function in the filter.

The Org export process has many filter sets applicable to different types of objects, plain text, parse trees, export options, and final output formats. The filters are named after the element type or object type: `org-export-filter-TYPE-functions`, where *TYPE* is the type targeted by the filter. Valid types are:

body	bold	babel-call
center-block	clock	code

diary-sexp	drawer	dynamic-block
entity	example-block	export-block
export-snippet	final-output	fixed-width
footnote-definition	footnote-reference	headline
horizontal-rule	inline-babel-call	inline-src-block
inlinetask	italic	item
keyword	latex-environment	latex-fragment
line-break	link	node-property
options	paragraph	parse-tree
plain-list	plain-text	planning
property-drawer	quote-block	radio-target
section	special-block	src-block
statistics-cookie	strike-through	subscript
superscript	table	table-cell
table-row	target	timestamp
underline	verbatim	verse-block

Here is an example filter that replaces non-breaking spaces ~ ~ in the Org buffer with '~' for the LaTeX backend.

```
(defun my-latex-filter-nobreaks (text backend info)
  "Ensure \" \" are properly handled in LaTeX export."
  (when (org-export-derived-backend-p backend 'latex)
    (replace-regexp-in-string " " "~" text)))

(add-to-list 'org-export-filter-plain-text-functions
             'my-latex-filter-nobreaks)
```

A filter requires three arguments: the code to be transformed, the name of the backend, and some optional information about the export process. The third argument can be safely ignored. Note the use of `org-export-derived-backend-p` predicate that tests for *latex* backend or any other backend, such as *beamer*, derived from *latex*.

Defining filters for individual files

The Org export can filter not just for backends, but also for specific files through the '`BIND`' keyword. Here is an example with two filters; one removes brackets from time stamps, and the other removes strike-through text. The filter functions are defined in a code block in the same Org file, which is a handy location for debugging.

```
#+BIND: org-export-filter-timestamp-functions (tmp-f-timestamp)
#+BIND: org-export-filter-strike-through-functions (tmp-f-strike-through)
#+BEGIN_SRC emacs-lisp :exports results :results none
  (defun tmp-f-timestamp (s backend info)
    (replace-regexp-in-string "&[lg]t;\\|[][]" "" s))
  (defun tmp-f-strike-through (s backend info) "")
#+END_SRC
```

Summary of the export process

Org mode export is a multi-step process that works on a temporary copy of the buffer. The export process consists of 4 major steps:

1. Process the temporary copy, making necessary changes to the buffer text;
2. Parse the buffer, converting plain Org markup into an abstract syntax tree (AST);
3. Convert the AST to text, as prescribed by the selected export backend;
4. Post-process the resulting exported text.

Process temporary copy of the source Org buffer [147]:

1. Execute `org-export-before-processing-functions` (see Export hooks);
2. Expand '`#+include`' keywords in the whole buffer (see Include Files);
3. Remove commented subtrees in the whole buffer (see Comment Lines);
4. Replace macros in the whole buffer (see Macro Replacement);
5. When `org-export-use-babel` is non-nil (default), process code blocks:
    - Leave code blocks inside archived subtrees (see Internal archiving) as is;
    - Evaluate all the other code blocks according to code block headers (see Limit code block evaluation);
    - Remove code, results of evaluation, both, or neither according to '`:exports`' header argument (see Exporting Code Blocks).

Parse the temporary buffer, creating AST:

1. Execute `org-export-before-parsing-functions` (see Export hooks). The hook functions may still modify the buffer;
2. Calculate export option values according to subtree-specific export settings, in-buffer keywords, '`#+BIND`' keywords, and buffer-local and global customization. The whole buffer is considered;
3. When `org-org-with-cite-processors` is non-nil (default), determine contributing bibliographies and record them into export options (see Citations). The whole buffer is considered;
4. Execute `org-export-filter-options-functions`;

5. Parse the accessible portion of the temporary buffer to generate an AST. The AST is a nested list of lists representing Org syntax elements (see Org Element API for more details):

   ```
   (org-data ...
    (heading
     (section
      (paragraph (plain-text) (bold (plain-text))))
     (heading)
     (heading (section ...))))
   ```

   Past this point, modifications to the temporary buffer no longer affect the export; Org export works only with the AST;

6. Remove elements that are not exported from the AST:
   - Headings according to 'SELECT_TAGS' and 'EXCLUDE_TAGS' export keywords; 'task', 'inline', 'arch' export options (see Export Settings);
   - Comments;
   - Clocks, drawers, fixed-width environments, footnotes, LaTeX environments and fragments, node properties, planning lines, property drawers, statistics cookies, timestamps, etc according to '#+OPTIONS' keyword (see Export Settings);
   - Table rows containing width and alignment markers, unless the selected export backend changes `:with-special-rows` export option to non-nil (see Column Width and Alignment);
   - Table columns containing recalc marks (see Advanced features).
7. Expand environment variables in file link AST nodes according to the 'expand-links' export option (see Export Settings);
8. Execute `org-export-filter-parse-tree-functions`. These functions can modify the AST by side effects;
9. When `org-org-with-cite-processors` is non-nil (default), replace citation AST nodes and '#+print_bibliography' keyword AST nodes as prescribed by the selected citation export processor (see Citation export processors).

Convert the AST to text by traversing the AST nodes, depth-first:

1. Convert the leaf nodes (without children) to text as prescribed by "transcoders" in the selected export backend [148];
2. Pass the converted nodes through the corresponding export filters (see Filters);
3. Concatenate all the converted child nodes to produce parent node contents;
4. Convert the nodes with children to text, passing the nodes themselves and their exported contents to the corresponding transcoders and then to the export filters (see Filters).

Post-process the exported text:

1. Post-process the converted AST, as prescribed by the export backend. [149] This step usually adds generated content (like Table of Contents) to the exported text;
2. Execute `org-export-filter-body-functions`;
3. Unless body-only export is selected (see The Export Dispatcher), add the necessary metadata to the final document, as prescribed by the export backend. Examples: Document author/title; HTML headers/footers; LaTeX preamble;

4. When `org-org-with-cite-processors` is non-nil (default), add bibliography metadata, as prescribed by the citation export processor;
5. Execute `org-export-filter-final-output-functions`.

Extending an existing backend

Some parts of the conversion process can be extended for certain elements so as to introduce a new or revised translation. That is how the HTML export backend was extended to handle Markdown format. The extensions work seamlessly so any aspect of filtering not done by the extended backend is handled by the original backend. Of all the export customization in Org, extending is very powerful as it operates at the parser level.

For this example, make the *ascii* backend display the language used in a source code block. Also make it display only when some attribute is non-`nil`, like the following:

```
#+ATTR_ASCII: :language t
```

Then extend ASCII backend with a custom “my-ascii” backend.

```
(defun my-ascii-src-block (src-block contents info)
  "Transcode a SRC-BLOCK element from Org to ASCII.
CONTENTS is nil.  INFO is a plist used as a communication
channel."
  (if (not (org-export-read-attribute :attr_ascii src-block :language))
      (org-export-with-backend 'ascii src-block contents info)
    (concat
     (format ",--[ %s ]--\n%s`----"
             (org-element-property :language src-block)
             (replace-regexp-in-string
              "^" "| "
              (org-element-normalize-string
               (org-export-format-code-default src-block info)))))))

(org-export-define-derived-backend 'my-ascii 'ascii
  :translate-alist '((src-block . my-ascii-src-block)))
```

The `my-ascii-src-block` function looks at the attribute above the current element. If not true, hands over to *ascii* backend. If true, which it is in this example, it creates a box around the code and leaves room for the inserting a string for language. The last form creates the new backend that springs to action only when translating `src-block` type elements.

To use the newly defined backend, evaluate the following from an Org buffer:

```
(org-export-to-buffer 'my-ascii "*Org MY-ASCII Export*")
```

Further steps to consider would be an interactive function, self-installing an item in the export dispatcher menu, and other user-friendly improvements. See https://orgmode.org/worg/dev/org-export-reference.html for more details.

13.18 Export Region

Some export backends include commands to convert a region of Org formatted text to another format, such as HTML or LaTeX. The conversion replaces the original source. Here are such commands:

`org-export-region-to-ascii` ¶
Convert the selected region into ASCII.

`org-export-region-to-utf8` ¶
Convert the selected region into UTF-8.

`org-export-region-to-html` ¶
Convert the selected region into HTML.

`org-export-region-to-latex` ¶
Convert the selected region into LaTeX.

`org-export-region-to-texinfo` ¶
Convert the selected region into Texinfo.

`org-export-region-to-md` ¶
Convert the selected region into Markdown.

The in-place conversion is particularly handy for quick conversion of tables and lists in non-Org buffers. For example, in an HTML buffer, write a list in Org syntax, select it, and convert it to HTML with `M-x org-html-convert-region-to-html`.

14 Publishing

Org includes a publishing management system that allows you to configure automatic HTML conversion of *projects* composed of interlinked Org files. You can also configure Org to automatically upload your exported HTML pages and related attachments, such as images and source code files, to a web server.

You can also use Org to convert files into PDF, or even combine HTML and PDF conversion so that files are available in both formats on the server.

Publishing has been contributed to Org by David O'Toole.

14.1 Configuration

Publishing needs significant configuration to specify files, destination and many other properties of a project.

- The variable `org-publish-project-alist`
- Sources and destinations for files
- Selecting files
- Publishing action
- Options for the exporters
- Publishing links
- Generating a sitemap
- Generating an index

14.1.1 The variable `org-publish-project-alist`

Publishing is configured almost entirely through setting the value of one variable, called `org-publish-project-alist`. Each element of the list configures one project, and may be in one of the two following forms:

```
("project-name" :property value :property value ...)
```

i.e., a well-formed property list with alternating keys and values, or:

```
("project-name" :components ("project-name" "project-name" ...))
```

In both cases, projects are configured by specifying property values. A project defines the set of files that are to be published, as well as the publishing configuration to use when publishing those files. When a project takes the second form listed above, the individual members of the `:components` property are taken to be sub-projects, which group together files requiring different publishing options. When you publish such a “meta-project”, all the components are also published, in the sequence given.

14.1.2 Sources and destinations for files

Most properties are optional, but some should always be set. In particular, Org needs to know where to look for source files, and where to put published files.

`:base-directory`
Directory containing publishing source files.

`:publishing-directory`
Directory where output files are published. You can directly publish to a webserver using a file name syntax appropriate for the Emacs tramp package. Or you can publish to a local directory and use external tools to upload your website (see Uploading Files).

`:preparation-function`
Function or list of functions to be called before starting the publishing process, for example, to run ‘`make`’ for updating files to be published. Each preparation function is called with a single argument, the project property list.

`:completion-function`
Function or list of functions called after finishing the publishing process, for example, to change permissions of the resulting files. Each completion function is called with a single argument, the project property list.

14.1.3 Selecting files

By default, all files with extension ‘`.org`’ in the base directory are considered part of the project. This can be modified by setting the following properties

`:base-extension`
Extension—without the dot—of source files. This actually is a regular expression. Set this to the symbol `any` if you want to get all files in `:base-directory`, even without extension.

`:exclude`
Regular expression to match file names that should not be published, even though they have been selected on the basis of their extension.

`:include`
List of files to be included regardless of `:base-extension` and `:exclude`.

`:recursive`
Non-`nil` means, check base-directory recursively for files to publish.

14.1.4 Publishing action

Publishing means that a file is copied to the destination directory and possibly transformed in the process. The default transformation is to export Org files as HTML files, and this is done by the function `org-html-publish-to-html` which calls the HTML exporter (see HTML Export). But you can also publish your content as PDF files using `org-latex-publish-to-pdf`, or as ASCII, Texinfo, etc., using the corresponding functions.

If you want to publish the Org file as an '`.org`' file but with *archived*, *commented*, and *tag-excluded* trees removed, use `org-org-publish-to-org`. This produces '`file.org`' and puts it in the publishing directory. If you want a htmlized version of this file, set the parameter `:htmlized-source` to `t`. It produces '`file.org.html`' in the publishing directory[150].

Other files like images only need to be copied to the publishing destination; for this you can use `org-publish-attachment`. For non-Org files, you always need to specify the publishing function:

`:publishing-function`
Function executing the publication of a file. This may also be a list of functions, which are all called in turn.

`:htmlized-source`
Non-`nil` means, publish htmlized source.

The function must accept three arguments: a property list containing at least a `:publishing-directory` property, the name of the file to be published, and the path to the publishing directory of the output file. It should take the specified file, make the necessary transformation, if any, and place the result into the destination folder.

Next: Publishing links, Previous: Publishing action, Up: Configuration [Contents][Index]

14.1.5 Options for the exporters

The property list can be used to set many export options for the HTML and LaTeX exporters. In most cases, these properties correspond to user variables in Org. The table below lists these properties along with the variable they belong to. See the documentation string for the respective variable for details.

When a property is given a value in `org-publish-project-alist`, its setting overrides the value of the corresponding user variable, if any, during publishing. Options set within a file (see Export Settings), however, override everything.

Generic properties

`:author`	`user-full-name`

:email	user-mail-address
:language	org-export-default-language
:select-tags	org-export-select-tags
:exclude-tags	org-export-exclude-tags
:creator	org-export-creator-string
:headline-levels	org-export-headline-levels
:preserve-breaks	org-export-preserve-breaks
:section-numbers	org-export-with-section-numbers
:time-stamp-file	org-export-timestamp-file
:with-archived-trees	org-export-with-archived-trees
:with-author	org-export-with-author
:expand-links	org-export-expand-links
:with-broken-links	org-export-with-broken-links
:with-clocks	org-export-with-clocks
:with-creator	org-export-with-creator
:with-date	org-export-with-date
:with-drawers	org-export-with-drawers
:with-email	org-export-with-email
:with-emphasize	org-export-with-emphasize
:with-entities	org-export-with-entities
:with-fixed-width	org-export-with-fixed-width
:with-footnotes	org-export-with-footnotes
:with-inlinetasks	org-export-with-inlinetasks
:with-latex	org-export-with-latex
:with-planning	org-export-with-planning
:with-priority	org-export-with-priority
:with-properties	org-export-with-properties
:with-smart-quotes	org-export-with-smart-quotes
:with-special-strings	org-export-with-special-strings
:with-statistics-cookies	org-export-with-statistics-cookies
:with-sub-superscript	org-export-with-sub-superscripts
:with-toc	org-export-with-toc

:with-tables	org-export-with-tables
:with-tags	org-export-with-tags
:with-tasks	org-export-with-tasks
:with-timestamps	org-export-with-timestamps
:with-title	org-export-with-title
:with-todo-keywords	org-export-with-todo-keywords
:with-cite-processors	org-export-process-citations
:cite-export	org-cite-export-processors

ASCII specific properties

:ascii-bullets	org-ascii-bullets
:ascii-caption-above	org-ascii-caption-above
:ascii-charset	org-ascii-charset
:ascii-global-margin	org-ascii-global-margin
:ascii-format-drawer-function	org-ascii-format-drawer-function
:ascii-format-inlinetask-function	org-ascii-format-inlinetask-function
:ascii-headline-spacing	org-ascii-headline-spacing
:ascii-indented-line-width	org-ascii-indented-line-width
:ascii-inlinetask-width	org-ascii-inlinetask-width
:ascii-inner-margin	org-ascii-inner-margin
:ascii-links-to-notes	org-ascii-links-to-notes
:ascii-list-margin	org-ascii-list-margin
:ascii-paragraph-spacing	org-ascii-paragraph-spacing
:ascii-quote-margin	org-ascii-quote-margin
:ascii-table-keep-all-vertical-lines	org-ascii-table-keep-all-vertical-lines
:ascii-table-use-ascii-art	org-ascii-table-use-ascii-art
:ascii-table-widen-columns	org-ascii-table-widen-columns
:ascii-text-width	org-ascii-text-width
:ascii-underline	org-ascii-underline
:ascii-verbatim-format	org-ascii-verbatim-format

Beamer specific properties

:headline-levels	org-beamer-frame-level

:beamer-subtitle-format	org-beamer-subtitle-format
:beamer-column-view-format	org-beamer-column-view-format
:beamer-theme	org-beamer-theme
:beamer-environments-extra	org-beamer-environments-extra
:beamer-frame-default-options	org-beamer-frame-default-options
:beamer-outline-frame-options	org-beamer-outline-frame-options
:beamer-outline-frame-title	org-beamer-outline-frame-title

HTML specific properties

:html-doctype	org-html-doctype
:html-container	org-html-container-element
:html-content-class	org-html-content-class
:html-html5-fancy	org-html-html5-fancy
:html-link-use-abs-url	org-html-link-use-abs-url
:html-link-home	org-html-link-home
:html-link-up	org-html-link-up
:html-equation-reference-format	org-html-equation-reference-format
:html-postamble	org-html-postamble
:html-preamble	org-html-preamble
:html-head	org-html-head
:html-head-extra	org-html-head-extra
:html-head-include-default-style	org-html-head-include-default-style
:html-head-include-scripts	org-html-head-include-scripts
:html-allow-name-attribute-in-anchors	org-html-allow-name-attribute-in-anchors
:html-divs	org-html-divs
:html-checkbox-type	org-html-checkbox-type
:html-extension	org-html-extension
:html-footnote-format	org-html-footnote-format
:html-footnote-separator	org-html-footnote-separator
:html-footnotes-section	org-html-footnotes-section
:html-format-drawer-function	org-html-format-drawer-function
:html-format-headline-function	org-html-format-headline-function

:html-format-inlinetask-function	org-html-format-inlinetask-function
:html-home/up-format	org-html-home/up-format
:html-indent	org-html-indent
:html-infojs-options	org-html-infojs-options
:html-infojs-template	org-html-infojs-template
:html-inline-image-rules	org-html-inline-image-rules
:html-link-org-files-as-html	org-html-link-org-files-as-html
:html-mathjax-options	org-html-mathjax-options
:html-mathjax-template	org-html-mathjax-template
:html-metadata-timestamp-format	org-html-metadata-timestamp-format
:html-postamble-format	org-html-postamble-format
:html-preamble-format	org-html-preamble-format
:html-prefer-user-labels	org-html-prefer-user-labels
:html-self-link-headlines	org-html-self-link-headlines
:html-table-align-individual-fields	org-html-table-align-individual-fields
:html-table-caption-above	org-html-table-caption-above
:html-table-data-tags	org-html-table-data-tags
:html-table-header-tags	org-html-table-header-tags
:html-table-use-header-tags-for-first-column	org-html-table-use-header-tags-for-first-column
:html-tag-class-prefix	org-html-tag-class-prefix
:html-text-markup-alist	org-html-text-markup-alist
:html-todo-kwd-class-prefix	org-html-todo-kwd-class-prefix
:html-toplevel-hlevel	org-html-toplevel-hlevel
:html-use-infojs	org-html-use-infojs
:html-validation-link	org-html-validation-link
:html-viewport	org-html-viewport
:html-inline-images	org-html-inline-images
:html-table-attributes	org-html-table-default-attributes
:html-table-row-open-tag	org-html-table-row-open-tag
:html-table-row-close-tag	org-html-table-row-close-tag
:html-xml-declaration	org-html-xml-declaration
:html-wrap-src-lines	org-html-wrap-src-lines

:html-klipsify-src	org-html-klipsify-src
:html-klipse-css	org-html-klipse-css
:html-klipse-js	org-html-klipse-js
:html-klipse-selection-script	org-html-klipse-selection-script
:creator	org-html-creator-string
:with-latex	org-html-with-latex

LaTeX specific properties

:latex-class	org-latex-default-class
:latex-active-timestamp-format	org-latex-active-timestamp-format
:latex-caption-above	org-latex-caption-above
:latex-classes	org-latex-classes
:latex-default-figure-position	org-latex-default-figure-position
:latex-default-table-environment	org-latex-default-table-environment
:latex-default-quote-environment	org-latex-default-quote-environment
:latex-default-table-mode	org-latex-default-table-mode
:latex-default-footnote-command	org-latex-default-footnote-command
:latex-diary-timestamp-format	org-latex-diary-timestamp-format
:latex-engraved-options	org-latex-engraved-options
:latex-engraved-preamble	org-latex-engraved-preamble
:latex-engraved-theme	org-latex-engraved-theme
:latex-footnote-defined-format	org-latex-footnote-defined-format
:latex-footnote-separator	org-latex-footnote-separator
:latex-format-drawer-function	org-latex-format-drawer-function
:latex-format-headline-function	org-latex-format-headline-function
:latex-format-inlinetask-function	org-latex-format-inlinetask-function
:latex-hyperref-template	org-latex-hyperref-template
:latex-image-default-scale	org-latex-image-default-scale
:latex-image-default-height	org-latex-image-default-height
:latex-image-default-option	org-latex-image-default-option
:latex-image-default-width	org-latex-image-default-width
:latex-images-centered	org-latex-images-centered

:latex-inactive-timestamp-format	org-latex-inactive-timestamp-format
:latex-inline-image-rules	org-latex-inline-image-rules
:latex-link-with-unknown-path-format	org-latex-link-with-unknown-path-format
:latex-src-block-backend	org-latex-src-block-backend
:latex-listings-langs	org-latex-listings-langs
:latex-listings-options	org-latex-listings-options
:latex-listings-src-omit-language	org-latex-listings-src-omit-language
:latex-minted-langs	org-latex-minted-langs
:latex-minted-options	org-latex-minted-options
:latex-prefer-user-labels	org-latex-prefer-user-labels
:latex-subtitle-format	org-latex-subtitle-format
:latex-subtitle-separate	org-latex-subtitle-separate
:latex-table-scientific-notation	org-latex-table-scientific-notation
:latex-tables-booktabs	org-latex-tables-booktabs
:latex-tables-centered	org-latex-tables-centered
:latex-text-markup-alist	org-latex-text-markup-alist
:latex-title-command	org-latex-title-command
:latex-toc-command	org-latex-toc-command
:latex-compiler	org-latex-compiler

Markdown specific properties

:md-footnote-format	org-md-footnote-format
:md-footnotes-section	org-md-footnotes-section
:md-headline-style	org-md-headline-style
:md-toplevel-hlevel	org-md-toplevel-hlevel

ODT specific properties

:odt-styles-file	org-odt-styles-file
:odt-content-template-file	org-odt-content-template-file
:odt-display-outline-level	org-odt-display-outline-level
:odt-fontify-srcblocks	org-odt-fontify-srcblocks
:odt-format-drawer-function	org-odt-format-drawer-function
:odt-format-headline-function	org-odt-format-headline-function

:odt-format-inlinetask-function	org-odt-format-inlinetask-function
:odt-inline-formula-rules	org-odt-inline-formula-rules
:odt-inline-image-rules	org-odt-inline-image-rules
:odt-pixels-per-inch	org-odt-pixels-per-inch
:odt-table-styles	org-odt-table-styles
:odt-use-date-fields	org-odt-use-date-fields
:with-latex	org-odt-with-latex

Texinfo specific properties

:texinfo-class	org-texinfo-default-class
:texinfo-classes	org-texinfo-classes
:texinfo-format-headline-function	org-texinfo-format-headline-function
:texinfo-node-description-column	org-texinfo-node-description-column
:texinfo-active-timestamp-format	org-texinfo-active-timestamp-format
:texinfo-inactive-timestamp-format	org-texinfo-inactive-timestamp-format
:texinfo-diary-timestamp-format	org-texinfo-diary-timestamp-format
:texinfo-link-with-unknown-path-format	org-texinfo-link-with-unknown-path-format
:texinfo-tables-verbatim	org-texinfo-tables-verbatim
:texinfo-table-scientific-notation	org-texinfo-table-scientific-notation
:texinfo-table-default-markup	org-texinfo-table-default-markup
:texinfo-text-markup-alist	org-texinfo-text-markup-alist
:texinfo-format-drawer-function	org-texinfo-format-drawer-function
:texinfo-format-inlinetask-function	org-texinfo-format-inlinetask-function
:texinfo-compact-itemx	org-texinfo-compact-itemx
:with-latex	org-texinfo-with-latex

14.1.6 Publishing links

To create a link from one Org file to another, you would use something like '`[[file:foo.org][The foo]]`' or simply '`[[file:foo.org]]`' (see External Links). When published, this link becomes a link to '`foo.html`'. You can thus interlink the pages of your "Org web" project and the links will work as expected when you publish them to HTML. If you also publish the Org source file and want to link to it, use an '`http`' link instead of a '`file:`' link, because '`file`' links are converted to link to the corresponding '`.html`' file.

Links to encrypted Org files, like '`[[file:foo.org.gpg]]`' are also supported.

You may also link to related files, such as images. Provided you are careful with relative file names, and provided you have also configured Org to upload the related files, these links will work too. See Example: complex publishing configuration, for an example of this usage.

Links between published documents can contain some search options (see Search Options in File Links), which will be resolved to the appropriate location in the linked file. For example, once published to HTML, the following links all point to a dedicated anchor in '`foo.html`'.

```
[[file:foo.org::*heading]]
[[file:foo.org::#custom-id]]
[[file:foo.org::target]]
```

14.1.7 Generating a sitemap

The following properties may be used to control publishing of a map of files for a given project.

`:auto-sitemap`
When non-`nil`, publish a sitemap during `org-publish-current-project` or `org-publish-all`.

`:sitemap-filename`
Filename for output of sitemap. Defaults to '`sitemap.org`', which becomes '`sitemap.html`'.

`:sitemap-title`
Title of sitemap page. Defaults to name of file.

`:sitemap-style`
Can be `list` (site-map is just an itemized list of the titles of the files involved) or `tree` (the directory structure of the source files is reflected in the site-map). Defaults to `tree`.

`:sitemap-format-entry` ¶
With this option one can tell how a site-map entry is formatted in the site-map. It is a function called with three arguments: the file or directory name relative to base directory of the project, the site-map style and the current project. It is expected to return a string. Default value turns file names into links and use document titles as descriptions. For specific formatting needs, one can use `org-publish-find-date`, `org-publish-find-title` and `org-publish-find-property`, to retrieve additional information about published documents.

`:sitemap-function`
Plug-in function to use for generation of the sitemap. It is called with two arguments: the title of the site-map and a representation of the files and directories involved in the project as a nested list, which can further be transformed using `org-list-to-generic`, `org-list-to-subtree` and alike. Default value generates a plain list of links to all files in the project.

`:sitemap-sort-folders`
Where folders should appear in the sitemap. Set this to `first` (default) or `last` to display folders first or last, respectively. When set to `ignore`, folders are ignored altogether. Any other value mixes files and folders. This variable has no effect when site-map style is `tree`.

`:sitemap-sort-files`
How the files are sorted in the site map. Set this to `alphabetically` (default), `chronologically` or `anti-chronologically`. `chronologically` sorts the files with older date first while `anti-chronologically` sorts the files with newer date first. `alphabetically` sorts the files

alphabetically. The date of a file is retrieved with `org-publish-find-date`.

`:sitemap-ignore-case`
Should sorting be case-sensitive? Default `nil`.

14.1.8 Generating an index

Org mode can generate an index across the files of a publishing project.

`:makeindex`
When non-`nil`, generate in index in the file '`theindex.org`' and publish it as '`theindex.html`'.

The file is created when first publishing a project with the `:makeindex` set. The file only contains a statement '`#+INCLUDE: "theindex.inc"`'. You can then build around this include statement by adding a title, style information, etc.

Index entries are specified with '`INDEX`' keyword. An entry that contains an exclamation mark creates a sub item.

```
*** Curriculum Vitae
#+INDEX: CV
#+INDEX: Application!CV
```

14.2 Uploading Files

For those people already utilizing third party sync tools such as Rsync or Unison, it might be preferable not to use the built-in remote publishing facilities of Org mode which rely heavily on Tramp. Tramp, while very useful and powerful, tends not to be so efficient for multiple file transfer and has been known to cause problems under heavy usage.

Specialized synchronization utilities offer several advantages. In addition to timestamp comparison, they also do content and permissions/attribute checks. For this reason you might prefer to publish your web to a local directory—possibly even *in place* with your Org files—and then use Unison or Rsync to do the synchronization with the remote host.

Since Unison, for example, can be configured as to which files to transfer to a certain remote destination, it can greatly simplify the project publishing definition. Simply keep all files in the correct location, process your Org files with `org-publish` and let the synchronization tool do the rest. You do not need, in this scenario, to include attachments such as JPG, CSS or PNG files in the project definition since the third-party tool syncs them.

Publishing to a local directory is also much faster than to a remote one, so that you can afford more easily to republish entire projects. If you set `org-publish-use-timestamps-flag` to `nil`, you gain the main benefit of re-including any changed external files such as source example files you might include with '`INCLUDE`' keyword. The timestamp mechanism in Org is not smart enough to detect if included files have been modified.

14.3 Sample Configuration

Below we provide two example configurations. The first one is a simple project publishing only a set of Org files. The second example is more complex, with a multi-component project.

- Example: simple publishing configuration
- Example: complex publishing configuration

14.3.1 Example: simple publishing configuration

This example publishes a set of Org files to the ‘`public_html`’ directory on the local machine.

```
(setq org-publish-project-alist
      '(("org"
         :base-directory "~/org/"
         :publishing-function org-html-publish-to-html
         :publishing-directory "~/public_html"
         :section-numbers nil
         :with-toc nil
         :html-head "<link rel=\"stylesheet\"
                     href=\"../other/mystyle.css\"
                     type=\"text/css\"/>")))
```

14.3.2 Example: complex publishing configuration

This more complicated example publishes an entire website, including Org files converted to HTML, image files, Emacs Lisp source code, and style sheets. The publishing directory is remote and private files are excluded.

To ensure that links are preserved, care should be taken to replicate your directory structure on the web server, and to use relative file paths. For example, if your Org files are kept in ‘`~/org/`’ and your publishable images in ‘`~/images/`’, you would link to an image with

```
file:../images/myimage.png
```

On the web server, the relative path to the image should be the same. You can accomplish this by setting up an ‘`images/`’ folder in the right place on the web server, and publishing images to it.

```
(setq org-publish-project-alist
      `(("orgfiles"
         :base-directory "~/org/"
         :base-extension "org"
         :publishing-directory "/ssh:user@host:~/html/notebook/"
         :publishing-function org-html-publish-to-html
         :exclude ,(rx (or "PrivateFile.org" (seq line-start "private/"))) ;; regexp
         :headline-levels 3
         :section-numbers nil
         :with-toc nil
         :html-head "<link rel=\"stylesheet\"
                 href=\"../other/mystyle.css\" type=\"text/css\"/>"
         :html-preamble t)

        ("images"
         :base-directory "~/images/"
         :base-extension "jpg\\|gif\\|png"
         :publishing-directory "/ssh:user@host:~/html/images/"
         :publishing-function org-publish-attachment)

        ("other"
         :base-directory "~/other/"
         :base-extension "css\\|el"
         :publishing-directory "/ssh:user@host:~/html/other/"
         :publishing-function org-publish-attachment)
        ("website" :components ("orgfiles" "images" "other"))))
```

14.4 Triggering Publication

Once properly configured, Org can publish with the following commands:

C-c C-e P x (**org-publish**) ¶
Prompt for a specific project and publish all files that belong to it.

C-c C-e P p (**org-publish-current-project**) ¶
Publish the project containing the current file.

C-c C-e P f (**org-publish-current-file**) ¶
Publish only the current file.

C-c C-e P a (**org-publish-all**) ¶
Publish every project.

Org uses timestamps to track when a file has changed. The above functions normally only publish changed files. You can override this and force publishing of all files by giving a prefix argument to any of the commands above, or by customizing the variable `org-publish-use-timestamps-flag`. This may be necessary in particular if files include other files via ‘`SETUPFILE`’ or ‘`INCLUDE`’ keywords.

15 Citation handling

While links (see Hyperlinks) are often sufficient to refer to external or internal information from Org, they have their limitations when referring to multiple targets or typesetting printed publications.

Org mode provides a more sophisticated markup to “cite” external resources. For example, consider the following Org mode snippet

```
#+bibliography: citationdata.bib

Org mode is used by various communities [cite:teaching: @orgteaching;
and TeX: @orgtex].  [cite/author/caps:@orgtex] uses Org mode to simplify
writing scientific publications, while [cite/author/caps:@orgteaching]
experiment with Org babel to improve teaching.

#+print_bibliography:
```

Org mode will gather citation metadata from the ‘`#+bibliography`’ database and use it to typeset the exported document in arbitrary formats. For example, the snippet below shows ASCII export output.

```
Org mode is used by various communities (teaching: Birkenkrahe, Marcus,
2023, and TeX: Somma, Emmanuele F, 2023).  Somma, Emmanuele F uses Org
mode to simplify writing scientific publications, while Birkenkrahe,
Marcus experiment with Org babel to improve teaching.

Birkenkrahe, Marcus (2023). /Teaching Data Science with Literate
Programming Tools/, MDPI.

Somma, Emmanuele F (2023). /Simplifying LaTeX with ORG-mode in Emacs/,
TUGboat volume.
```

In addition to export, users can use completion to search and insert citations from the bibliography (via `org-cite-insert`). Citations also act like ordinary links, jumping to the citation metadata when “following” them using `org-open-at-point`.

You can customize every aspect (*capability*) of citation handling using built-in or external *citation processors*.

Org mode ships with several built-in citation processors tailored to work with LaTeX export and BibTeX bibliographies (‘`bibtex`’, ‘`biblatex`’, and ‘`natbib`’ processors), or with more generic formats described using Citation Style Language (‘`csl`’ processor). The default citation processor is ‘`basic`’ - it works with arbitrary export formats and recognizes both BibTeX and CSL bibliographies. More citation processors are distributed as Emacs packages.

Multiple citation processors can be mixed to meet your preferences. Configure `org-cite-activate-processor`, `org-cite-follow-processor`, `org-cite-insert-processor`, and `org-cite-export-processors` to select which processor to use for every citation capability:

activate
Fontification, tooltip preview, etc.

follow
At-point actions on citations via `org-open-at-point`.

insert
Add and edit citations via `org-cite-insert`.

export

Via different libraries for different target formats.

15.1 Citations

Before adding citations, first set one-or-more bibliographies, either globally with `org-cite-global-bibliography`, or locally using one or more “bibliography” keywords.

```
#+bibliography: SomeFile.bib
#+bibliography: /some/other/file.json
#+bibliography: "/some/file/with spaces/in its name.bib"
```

Org mode uses all the local and global bibliographies combined to search for citation keys.

One can then insert and edit citations using `org-cite-insert`, called with `C-c C-x @`.

A *citation* requires one or more citation *key(s)*, elements identifying a reference in the bibliography.

- Each citation is surrounded by brackets and uses the ‘`cite`’ type.
- Each key starts with the character ‘@’.

  ```
  [cite:@key]
  ```

- Each key can be qualified by a *prefix* (e.g. “see ”) and/or a *suffix* (e.g. “p. 123”), giving information useful or necessary for the comprehension of the citation but not included in the reference.

  ```
  [cite:see @key p. 123]
  ```

- A single citation can cite more than one reference ; the keys are separated by semicolons ; the formatting of such citation groups is specified by the style.

  ```
  [cite:@key1;@key2;@key3]
  ```

- One can also specify a stylistic variation for the citations by inserting a ‘`/`’ and a style name between the ‘`cite`’ keyword and the colon; this usually makes sense only for the author-year styles.

  ```
  [cite/style:common prefix ;prefix @key suffix; ... ; common suffix]
  ```

  When ‘`style`’ is not specified, one of the two default styles are used

  - either the default style specified in the ‘`CITE_EXPORT`’ keyword (see Citation export processors)

    ```
    #+cite_export: basic numeric noauthor/bare
    [cite:@key] is the same as [cite/noauthor/bare:@key]
    ```

  - or, if ‘`CITE_EXPORT`’ is not set, using the default ‘`nil`’ style

    ```
    [cite:@key] is the same as [cite/nil:@key]
    ```

The only mandatory elements are:

- The ‘`cite`’ keyword and the colon.

- The '@' character immediately preceding each key.
- The brackets surrounding the citation(s) (group).

15.2 Citation export processors

Org currently includes the following export processors:

- Two processors can export to a variety of formats, including '`latex`' (and therefore '`pdf`'), '`html`', '`odt`' and plain (UTF8) text:
  basic
  a basic export processor, well adapted to situations where backward compatibility is not a requirement and formatting needs are minimal;

  csl
  this export processor uses format files written in Citation Style Language via citeproc-el;

- In contrast, three other processors target LaTeX and LaTeX-derived formats exclusively:
  bibtex
  this export processor uses BibTeX, the historical bibliographic processor used with LaTeX, thus allowing the use of data and style files compatible with this processor (including a large number of publishers' styles). It only supports LaTeX's '`\cite`' and '`\nocite`' commands.

  natbib
  as with the bibtex processor, but using the LaTeX package '`natbib`', allowing more stylistic variants that LaTeX's '`\cite`' command.

  biblatex
  this backend allows the use of data and formats prepared for BibLaTeX, an alternate bibliographic processor used with LaTeX, which overcomes some serious BibTeX limitations, but has not (yet?) been widely adopted by publishers.

The '`CITE_EXPORT`' keyword specifies the export processor, citation style, and bibliography style; for example (all arguments are optional)

```
#+cite_export: [export processor name] [bibliography style] [default citation style]
#+cite_export: basic author-year author
```

specifies the '`basic`' export processor with citations inserted as author's name and references indexed by author's names and year;

```
#+cite_export: csl /some/path/to/vancouver-brackets.csl
```

specifies the '`csl`' processor and CSL style, which in this case defines numeric citations and numeric references according to the '`Vancouver`' specification (as style used in many medical journals), following a typesetting variation putting citations between brackets;

```
#+cite_export: natbib kluwer
```

specifies the '`natbib`' export processor with a label citation style conformant to the Harvard style and the specification of the Wolkers-Kluwer publisher; since it relies on the `bibtex` processor of your LaTeX installation, it won't export to anything but PDF.

```
#+cite_export: biblatex numeric,backend=bibtex
```

specifies the ‘biblatex’ export processor with the default ‘numeric’ style and the ‘bibtex’ backend. Always define the style first and then the rest of load-time options for the ‘biblatex’ package. Alternatively, you can use the key=val,key=val format for the options as documented in the ‘biblatex’ package documentation:

```
#+cite_export: biblatex backend=bibtex,style=numeric
```

The org-cite-biblatex-options variable in your Emacs configuration uses this format. It will only export to PDF, since it relies on the biblatex processor of your LaTeX installation.

15.3 Bibliography printing

The ‘PRINT_BIBLIOGRAPHY’ keyword specifies where the bibliography should be printed (note the colon):

```
#+print_bibliography:
```

The bibliography printed by the LaTeX-based export processors ‘bibtex’, ‘natbib’ and ‘biblatex’ has a chapter or section heading by default, while the ‘basic’ and ‘csl’ processors print the list of bibliography entries without a heading.

A document may contain more than one ‘PRINT_BIBLIOGRAPHY’ keywords. Each of the keywords will trigger printing the bibliography.

The keywords can be used with or without additional options. Options can be used, for example, to print only entries that belong to a certain category or to control formatting. The set of supported ‘PRINT_BIBLIOGRAPHY’ options and their interpretation varies between the different citation export processors. Some export processors do not support passing options.

Bibliography options in the ‘biblatex’ and ‘csl’ export processors

15.3.1 Bibliography options in the ‘biblatex’ and ‘csl’ export processors

The ‘biblatex’ and ‘csl’ export processors support bibliography options through a property list attached to the ‘PRINT_BIBLIOGRAPHY’ keyword. For example,

```
#+print_bibliography: :keyword algebra :type book
```

Values including spaces must be surrounded with double quotes. If you need to use a key multiple times, you can separate its values with commas, but without any space in-between:

```
#+print_bibliography: :keyword "algebraic logic" :nottype article,book
```

The ‘biblatex’ export processor accepts all options supported by BibLaTeX’s \printbibliography command.

The ‘csl’ processor accepts the following options:

‘:keyword <keyword(,keyword2...)>’
Print only entries whose keyword field contains all given keywords.

‘:notkeyword <keyword(,keyword2...)>’
Print only entries whose keyword field does not contain any of the given keywords.

‘:type <entrytype>’
Print only entries whose type is ‘<entrytype>’. Entry type is the BibTeX/BibLaTeX entry type if this information is available (the entry was read from a BibTeX/BibLaTeX bibliography) and the CSL entry type otherwise.

‘:nottype <entrytype(,entrytype2...)>’
Print only entries whose type is not among the given entry types. Entry type is determined as in the case of ‘:type’.

‘:csltype <entrytype>’
Print only entries whose CSL entry type (possibly based on a conversion from BibTeX/BibLaTeX to CSL) is ‘<entrytype>’.

‘:notcsltype <entrytype(,entrytype2...)>’
Print only entries whose CSL entry type (possibly based on a conversion from BibTeX/BibLaTeX to CSL) is not among the listed entry types.

‘:filter <predicate>’
Print only entries for which the given Emacs Lisp predicate returns a non-nil value.

Next: Miscellaneous, Previous: Citation handling, Up: The Org Manual [Contents][Index]

16 Working with Source Code

Source code here refers to any plain text collection of computer instructions, possibly with comments, written using a human-readable programming language. Org can manage source code in an Org document when the source code is identified with begin and end markers. Working with source code begins with identifying source code blocks. A source code block can be placed almost anywhere in an Org document; it is not restricted to the preamble or the end of the document. However, Org cannot manage a source code block if it is placed inside an Org comment or within a fixed width section.

Here is an example source code block in the Emacs Lisp language:

```
#+BEGIN_SRC emacs-lisp
  (defun org-xor (a b)
     "Exclusive or."
     (if a (not b) b))
#+END_SRC
```

Source code blocks are one of many Org block types, which also include “center”, “comment”, “dynamic”, “example”, “export”, “quote”, “special”, and “verse”. This section pertains to blocks between ‘#+BEGIN_SRC’ and ‘#+END_SRC’.

Details of Org’s facilities for working with source code are described in the following sections.

16.1 Features Overview

Org can manage the source code in the block delimited by '`#+BEGIN_SRC`' … '`#+END_SRC`' in several ways that can simplify housekeeping tasks essential to modern source code maintenance. Org can edit, format, extract, export, and publish source code blocks. Org can also compile and execute a source code block, then capture the results. The Org mode literature sometimes refers to source code blocks as *live code* blocks because they can alter the content of the Org document or the material that it exports. Users can control the "liveliness" of each source code block by tweaking the header arguments (see Using Header Arguments) for compiling, execution, extraction, and exporting.

For editing and formatting a source code block, Org uses an appropriate Emacs major mode that includes features specifically designed for source code in that language.

Org can extract one or more source code blocks and write them to one or more source files—a process known as *tangling* in literate programming terminology.

For exporting and publishing, Org's backends can format a source code block appropriately, often with native syntax highlighting.

For executing and compiling a source code block, the user can configure Org to select the appropriate compiler. Org provides facilities to collect the result of the execution or compiler output, insert it into the Org document, and/or export it. In addition to text results, Org can insert links to other data types, including audio, video, and graphics. Org can also link a compiler error message to the appropriate line in the source code block.

An important feature of Org's management of source code blocks is the ability to pass variables, functions, and results to one another using a common syntax for source code blocks in any language. Although most literate programming facilities are restricted to one language or another, Org's language-agnostic approach lets the literate programmer match each programming task with the appropriate computer language and to mix them all together in a single Org document. This interoperability among languages explains why Org's source code management facility was named *Org Babel* by its originators, Eric Schulte and Dan Davison.

Org mode fulfills the promise of easy verification and maintenance of publishing reproducible research by keeping text, data, code, configuration settings of the execution environment, the results of the execution, and associated narratives, claims, references, and internal and external links in a single Org document.

16.2 Structure of Code Blocks

Org offers two ways to structure source code in Org documents: in a source code block, and directly inline. Both specifications are shown below.

A source code block conforms to this structure:

```
#+NAME: <name>
#+BEGIN_SRC <language> <switches> <header arguments>
  <body>
#+END_SRC
```

Do not be put-off by having to remember the source block syntax. Org mode offers a command for wrapping existing text in a block (see Structure Templates). Org also works with other completion systems in Emacs, some of which predate Org and have custom domain-specific languages for defining templates. Regular use of templates reduces errors, increases accuracy, and maintains consistency.

An inline code block conforms to this structure:

```
src_<language>{<body>}
```

or

```
src_<language>[<header arguments>]{<body>}
```

‘#+NAME: <name>’
Optional. Names the source block so it can be called, like a function, from other source blocks or inline code to evaluate or to capture the results. Code from other blocks, other files, and from table formulas (see The Spreadsheet) can use the name to reference a source block. This naming serves the same purpose as naming Org tables. Org mode requires unique names. For duplicate names, Org mode’s behavior is undefined. Inline code blocks cannot have a name.

‘#+BEGIN_SRC’ … ‘#+END_SRC’
Mandatory. They mark the start and end of a block that Org requires. The ‘`#+BEGIN_SRC`’ line takes additional arguments, as described next.

‘<language>’ ¶
Optional. It is the identifier of the source code language in the block. See Languages, for identifiers of supported languages.

When ‘`<language>`’ identifier is omitted, the block also cannot have ‘`<switches>`’ and ‘`<header arguments>`’.

Language identifier is also used to fontify code blocks in Org buffers, when `org-src-fontify-natively` is set to non-`nil`. See Editing Source Code.

‘<switches>’ ¶
Optional. Switches provide finer control of the code execution, export, and format (see the discussion of switches in Literal Examples).

Optional. Heading arguments control many aspects of evaluation, export and tangling of code blocks (see Using Header Arguments). Using Org’s properties feature, header arguments can be selectively applied to the entire buffer or specific subtrees of the Org document.

‘<body>’
Source code in the dialect of the specified language identifier.

Next: Environment of a Code Block, Previous: Structure of Code Blocks, Up: Working with Source Code [Contents][Index]

16.3 Using Header Arguments

Org comes with many header arguments common to all languages. New header arguments are added for specific languages as they become available for use in source code blocks. A header argument is specified with an initial colon followed by the argument's name in lowercase.

Since header arguments can be set in several ways, Org prioritizes them in case of overlaps or conflicts by giving local settings a higher priority. Header values in function calls, for example, override header values from global defaults.

System-wide header arguments

System-wide values of header arguments can be specified by customizing the `org-babel-default-header-args` variable, which defaults to the following values:

```
:session    => "none"
:results    => "replace"
:exports    => "code"
:cache      => "no"
:noweb      => "no"
:hlines     => "no"
:tangle     => "no"
```

Inline source blocks (see Structure of Code Blocks) use slightly different default header arguments defined in `org-babel-default-inline-header-args`:

```
:session    => "none"
:results    => "replace"
:exports    => "results"
:hlines     => "yes"
```

The most notable difference between default header arguments for inline and normal source blocks is '`:exports`' argument. For inline source blocks, results of evaluation are exported by default; not the code.

Unlike the default values, header arguments set using Org mode properties (see Header arguments in Org mode properties) do apply to both the normal source blocks and inline source blocks.

The example below sets '`:noweb`' header arguments to '`yes`', which makes Org expand '`:noweb`' references by default.

```
(setq org-babel-default-header-args
      (cons '(:noweb . "yes")
            (assq-delete-all :noweb org-babel-default-header-args)))
```

Each language can have separate default header arguments by customizing the variable `org-babel-default-header-args:<LANG>`, where *<LANG>* is the name of the language. For details, see the language-specific online documentation at https://orgmode.org/worg/org-contrib/babel/.

Header arguments in Org mode properties

For header arguments applicable to the buffer, use '`PROPERTY`' keyword anywhere in the Org file (see Property Syntax).

The following example makes all the R code blocks execute in the same session. Setting ‘`:results`’ to ‘`silent`’ ignores the results of executions for all blocks, not just R code blocks; no results inserted for any block.

```
#+PROPERTY: header-args:R  :session *R*
#+PROPERTY: header-args    :results silent
```

Header arguments set through Org’s property drawers (see Property Syntax) apply at the subtree level on down. Since these property drawers can appear anywhere in the file hierarchy, Org uses outermost call or source block to resolve the values. Org ignores `org-use-property-inheritance` setting.

In this example, ‘`:cache`’ defaults to ‘`yes`’ for all code blocks in the subtree.

```
* sample header
  :PROPERTIES:
  :header-args:    :cache yes
  :END:
```

Properties defined through `org-set-property` function, bound to `C-c C-x p`, apply to all active languages. They override properties set in `org-babel-default-header-args`.

Language-specific header arguments are also read from properties ‘`header-args:<LANG>`’ where *<LANG>* is the language identifier. For example,

```
* Heading
  :PROPERTIES:
  :header-args:clojure:    :session *clojure-1*
  :header-args:R:          :session *R*
  :END:
** Subheading
  :PROPERTIES:
  :header-args:clojure:    :session *clojure-2*
  :END:
```

would force separate sessions for Clojure blocks in ‘`Heading`’ and ‘`Subheading`’, but use the same session for all R blocks. Blocks in ‘`Subheading`’ inherit settings from ‘`Heading`’.

Code block specific header arguments

Header arguments are most commonly set at the source code block level, on the ‘`#+BEGIN_SRC`’ line. Arguments set at this level take precedence over those set in the `org-babel-default-header-args` variable, and also those set as header properties.

In the following example, setting ‘`:results`’ to ‘`silent`’ makes it ignore results of the code execution. Setting ‘`:exports`’ to ‘`code`’ exports only the body of the code block to HTML or LaTeX.

```
#+NAME: factorial
#+BEGIN_SRC haskell :results silent :exports code :var n=0
  fac 0 = 1
  fac n = n * fac (n-1)
#+END_SRC
```

The same header arguments in an inline code block:

```
src_haskell[:exports both]{fac 5}
```

Code block header arguments can span multiple lines using ‘#+HEADER:’ on each line. Note that Org currently accepts the plural spelling of ‘#+HEADER:’ only as a convenience for backward-compatibility. It may be removed at some point.

Multi-line header arguments on an unnamed code block:

```
#+HEADER: :var data1=1
#+BEGIN_SRC emacs-lisp :var data2=2
   (message "data1:%S, data2:%S" data1 data2)
#+END_SRC

#+RESULTS:
: data1:1, data2:2
```

Multi-line header arguments on a named code block:

```
#+NAME: named-block
#+HEADER: :var data=2
#+BEGIN_SRC emacs-lisp
  (message "data:%S" data)
#+END_SRC

#+RESULTS: named-block
  : data:2
```

Header arguments in function calls

Header arguments in function calls are the most specific and override all other settings in case of an overlap. They get the highest priority. Two ‘#+CALL:’ examples are shown below. For the complete syntax of ‘CALL’ keyword, see Evaluating Code Blocks.

In this example, ‘:exports results’ header argument is applied to the evaluation of the ‘#+CALL:’ line.

```
#+CALL: factorial(n=5) :exports results
```

In this example, ‘:session special’ header argument is applied to the evaluation of ‘factorial’ code block.

```
#+CALL: factorial[:session special](n=5)
```

Next: Evaluating Code Blocks, Previous: Using Header Arguments, Up: Working with Source Code [Contents][Index]

16.4 Environment of a Code Block

Passing arguments

Use ‘var’ for passing arguments to source code blocks. The specifics of variables in code blocks vary by the source language and are covered in the language-specific documentation. The syntax for ‘var’, however, is the same for all languages. This includes declaring a variable, and assigning a default value.

The following syntax is used to pass arguments to code blocks using the ‘`var`’ header argument.

```
:var NAME=ASSIGN
```

NAME is the name of the variable bound in the code block body. *ASSIGN* is a literal value, such as a string, a number, a reference to a table, a list, a literal example, another code block—with or without arguments—or the results of evaluating a code block. *ASSIGN* may specify a filename for references to elements in a different file, using a ‘`:`’ to separate the filename from the reference.

```
:var NAME=FILE:REFERENCE
```

When ‘`FILE`’ does not exist, the reference is searched in the current file, using the verbatim reference. This way, ‘`:var table=tbl:example`’ will be searched inside the current buffer.

Here are examples of passing values by reference:

table
A table named with a ‘`NAME`’ keyword.

```
#+NAME: example-table
| 1 |
| 2 |
| 3 |
| 4 |

#+NAME: table-length
#+BEGIN_SRC emacs-lisp :var table=example-table
  (length table)
#+END_SRC

#+RESULTS: table-length
: 4
```

When passing a table, you can treat specially the row, or the column, containing labels for the columns, or the rows, in the table.

The ‘`colnames`’ header argument accepts ‘`yes`’, ‘`no`’, or ‘`nil`’ values. The default value is ‘`nil`’: if an input table has column names—because the second row is a horizontal rule—then Org removes the column names, processes the table, puts back the column names, and then writes the table to the results block. Using ‘`yes`’, Org does the same to the first non-hline row, even if the initial table does not contain any horizontal rule. When set to ‘`no`’, Org does not pre-process column names at all.

```
#+NAME: less-cols
| a |
|---|
| b |
| c |

#+BEGIN_SRC python :var tab=less-cols :colnames nil
return [[val + '*' for val in row] for row in tab]
#+END_SRC

#+RESULTS:
| a  |
|----|
| b* |
| c* |
```

Similarly, the ‘`rownames`’ header argument can take two values: ‘`yes`’ or ‘`no`’. When set to ‘`yes`’, Org removes the first column, processes the table, puts back the first column, and then writes the table to the results block. The default is ‘`no`’, which means Org does not pre-process the first column. Note that Emacs Lisp code blocks ignore ‘`rownames`’ header argument because of the ease of table-handling in Emacs.

```
#+NAME: with-rownames
| one | 1 | 2 | 3 | 4 |  5 |
| two | 6 | 7 | 8 | 9 | 10 |

#+BEGIN_SRC python :var tab=with-rownames :rownames yes
return [[val + 10 for val in row] for row in tab]
#+END_SRC

#+RESULTS:
| one | 11 | 12 | 13 | 14 | 15 |
| two | 16 | 17 | 18 | 19 | 20 |
```

To refer to a table in another file, join the filename and table name with a colon, for example: ‘`:var table=other-file.org:example-table`’.

list

A simple named list.

```
#+NAME: example-list
- simple
  - not
  - nested
- list

#+BEGIN_SRC emacs-lisp :var x=example-list
  (print x)
#+END_SRC

#+RESULTS:
| simple | list |
```

Note that only the top level list items are passed along. Nested list items are ignored.

code block without arguments

A code block name, as assigned by ‘NAME’ keyword from the example above, optionally followed by parentheses.

```
#+BEGIN_SRC emacs-lisp :var length=table-length()
  (* 2 length)
#+END_SRC

#+RESULTS:
: 8
```

code block with arguments

A code block name, as assigned by ‘NAME’ keyword, followed by parentheses and optional arguments passed within the parentheses. The block is evaluated with point at its location.

```
#+NAME: double
#+BEGIN_SRC emacs-lisp :var input=8
  (* 2 input)
#+END_SRC

#+RESULTS: double
: 16

#+NAME: squared
#+BEGIN_SRC emacs-lisp :var input=double(input=1)
  (* input input)
#+END_SRC

#+RESULTS: squared
: 4
```

literal example, or code block contents

A code block or literal example block named with a ‘NAME’ keyword, followed by brackets (optional for example blocks).

```
#+NAME: literal-example
#+BEGIN_EXAMPLE
  A literal example
  on two lines
#+END_EXAMPLE

#+NAME: read-literal-example
#+BEGIN_SRC emacs-lisp :var x=literal-example[]
  (concatenate #'string x " for you.")
#+END_SRC

#+RESULTS: read-literal-example
: A literal example
: on two lines for you.
```

Indexing variable values enables referencing portions of a variable. Indexes are 0 based with negative values counting backwards from the end. If an index is separated by commas then each subsequent section indexes as the next dimension. Note that this indexing occurs *before* other table-related header arguments are applied, such as ‘`hlines`’, ‘`colnames`’ and ‘`rownames`’. The following example assigns the last cell of the first row the table ‘`example-table`’ to the variable ‘`data`’:

```
#+NAME: example-table
| 1 | a |
| 2 | b |
| 3 | c |
| 4 | d |

#+BEGIN_SRC emacs-lisp :var data=example-table[0,-1]
  data
#+END_SRC

#+RESULTS:
: a
```

Two integers separated by a colon reference a range of variable values. In that case the entire inclusive range is referenced. For example the following assigns the middle three rows of ‘`example-table`’ to ‘`data`’.

```
#+NAME: example-table
| 1 | a |
| 2 | b |
| 3 | c |
| 4 | d |
| 5 | 3 |

#+BEGIN_SRC emacs-lisp :var data=example-table[1:3]
  data
#+END_SRC

#+RESULTS:
| 2 | b |
| 3 | c |
| 4 | d |
```

To pick the entire range, use an empty index, or the single character ‘`*`’. ‘`0:-1`’ does the same thing. Example below shows how to reference the first column only.

```
#+NAME: example-table
| 1 | a |
| 2 | b |
| 3 | c |
| 4 | d |

#+BEGIN_SRC emacs-lisp :var data=example-table[,0]
  data
#+END_SRC

#+RESULTS:
| 1 | 2 | 3 | 4 |
```

Index referencing can be used for tables and code blocks. Index referencing can handle any number of dimensions. Commas delimit multiple dimensions, as shown below.

```
#+NAME: 3D
#+BEGIN_SRC emacs-lisp
  '(((1  2  3)  (4  5  6)  (7  8  9))
    ((10 11 12) (13 14 15) (16 17 18))
    ((19 20 21) (22 23 24) (25 26 27)))
#+END_SRC

#+BEGIN_SRC emacs-lisp :var data=3D[1,,1]
  data
#+END_SRC

#+RESULTS:
| 11 | 14 | 17 |
```

Note that row names and column names are not removed prior to variable indexing. You need to take them into account, even when ‘`colnames`’ or ‘`rownames`’ header arguments remove them.

Emacs lisp code can also set the values for variables. To differentiate a value from Lisp code, Org interprets any value starting with ‘(’, ‘[’, ‘'’ or ‘`’ as Emacs Lisp code. The result of evaluating that code is then assigned to the value of that variable. The following example shows how to reliably query and pass the file name of the Org mode buffer to a code block using headers. We need reliability here because the file’s name could change once the code in the block starts executing.

```
#+BEGIN_SRC sh :var filename=(buffer-file-name) :exports both
  wc -w $filename
#+END_SRC
```

Note that values read from tables and lists are not mistakenly evaluated as Emacs Lisp code, as illustrated in the following example.

```
#+NAME: table
| (a b c) |

#+HEADER: :var data=table[0,0]
#+BEGIN_SRC perl
  $data
#+END_SRC

#+RESULTS:
: (a b c)
```

Using sessions

Two code blocks can share the same environment. The ‘`session`’ header argument is for running multiple source code blocks under one session. Org runs code blocks with the same session name in the same interpreter process.

‘none’
Default. Each code block gets a new interpreter process to execute. The process terminates once the block is evaluated.

STRING

Any string besides ‘none’ turns that string into the name of that session. For example, ‘`:session STRING`’ names it ‘`STRING`’. If ‘`session`’ has no value, then the session name is derived from the source language identifier. Subsequent blocks with the same source code language use the same session. Depending on the language, state variables, code from other blocks, and the overall interpreted environment may be shared. Some interpreted languages support concurrent sessions when subsequent source code language blocks change session names.

Only languages that provide interactive evaluation can have session support. Not all languages provide this support, such as C and ditaa. Even languages, such as Python and Haskell, that do support interactive evaluation impose limitations on allowable language constructs that can run interactively. Org inherits those limitations for those code blocks running in a session.

Choosing a working directory

The ‘`dir`’ header argument specifies the default directory during code block execution. If it is absent, then the directory associated with the current buffer is used. In other words, supplying ‘`:dir DIRECTORY`’ temporarily has the same effect as changing the current directory with `M-x cd RET DIRECTORY`, and then not setting ‘`dir`’. Under the surface, ‘`dir`’ simply sets the value of the Emacs variable `default-directory`. Setting ‘`mkdirp`’ header argument to a non-`nil` value creates the directory, if necessary.

Setting ‘`dir`’ to the symbol `attach` or the string `"'attach"` will set ‘`dir`’ to the directory returned by `(org-attach-dir)`, set ‘`:mkdir yes`’, and insert any file paths, as when using ‘`:results file`’, which are under the node’s attachment directory using ‘`attachment:`’ links instead of the usual ‘`file:`’ links. Any returned path outside of the attachment directory will use ‘`file:`’ links as per usual.

For example, to save the plot file in the ‘`Work/`’ folder of the home directory—notice tilde is expanded:

```
#+BEGIN_SRC R :file myplot.png :dir ~/Work
  matplot(matrix(rnorm(100), 10), type="l")
#+END_SRC
```

To evaluate the code block on a remote machine, supply a remote directory name using Tramp syntax. For example:

```
#+BEGIN_SRC R :file plot.png :dir /scp:dand@yakuba.princeton.edu:
  plot(1:10, main=system("hostname", intern=TRUE))
#+END_SRC
```

Org first captures the text results as usual for insertion in the Org file. Then Org also inserts a link to the remote file, thanks to Emacs Tramp. Org constructs the remote path to the file name from ‘`dir`’ and `default-directory`, as illustrated here:

```
[[file:/scp:dand@yakuba.princeton.edu:/home/dand/plot.png][plot.png]]
```

When ‘`dir`’ is used with ‘`session`’, Org sets the starting directory for a new session. But Org does not alter the directory of an already existing session.

Do not use ‘dir’ with ‘:exports results’ or with ‘:exports both’ to avoid Org inserting incorrect links to remote files. That is because Org does not expand `default directory` to avoid some underlying portability issues.

Inserting headers and footers

The ‘prologue’ header argument is for appending to the top of the code block for execution, like a reset instruction. For example, you may use ‘:prologue "reset"’ in a Gnuplot code block or, for every such block:

```
(add-to-list 'org-babel-default-header-args:gnuplot
             '((:prologue . "reset")))
```

Likewise, the value of the ‘epilogue’ header argument is for appending to the end of the code block for execution.

Next: Results of Evaluation, Previous: Environment of a Code Block, Up: Working with Source Code [Contents][Index]

16.5 Evaluating Code Blocks

A note about security: With code evaluation comes the risk of harm. Org safeguards by prompting for user’s permission before executing any code in the source block. To customize this safeguard, or disable it, see Code Evaluation and Security Issues.

How to evaluate source code

Org captures the results of the code block evaluation and inserts them in the Org file, right after the code block. The insertion point is after a newline and the ‘RESULTS’ keyword. Org creates the ‘RESULTS’ keyword if one is not already there. More details in Results of Evaluation.

By default, Org enables only Emacs Lisp code blocks for execution. See Languages to enable other languages.

Org provides many ways to execute code blocks. `C-c C-c` or `C-c C-v e` with the point on a code block[151] calls the `org-babel-execute-src-block` function, which executes the code in the block, collects the results, and inserts them in the buffer.

By calling a named code block[152] from an Org mode buffer or a table. Org can call the named code blocks from the current Org mode buffer or from the “Library of Babel” (see Library of Babel).

The syntax for ‘CALL’ keyword is:

```
#+CALL: <name>(<arguments>)
#+CALL: <name>[<inside header arguments>](<arguments>) <end header arguments>
```

The syntax for inline named code blocks is:

```
... call_<name>(<arguments>) ...
... call_<name>[<inside header arguments>](<arguments>)[<end header arguments>] ...
```

When inline syntax is used, the result is wrapped based on the variable `org-babel-inline-result-wrap`, which by default is set to `"=%s="` to produce verbatim text suitable for markup.

‘<name>’
This is the name of the code block (see Structure of Code Blocks) to be evaluated in the current document. If the block is located in another file, start ‘<name>’ with the file name followed by a colon. For example, in order to execute a block named ‘clear-data’ in ‘file.org’, you can write the following:

```
#+CALL: file.org:clear-data()
```

‘<arguments>’
Org passes arguments to the code block using standard function call syntax. For example, a ‘#+CALL:’ line that passes ‘4’ to a code block named ‘double’, which declares the header argument ‘:var n=2’, would be written as:

```
#+CALL: double(n=4)
```

Note how this function call syntax is different from the header argument syntax.

‘<inside header arguments>’
Org passes inside header arguments to the named code block using the header argument syntax. Inside header arguments apply to code block evaluation. For example, ‘[:results output]’ collects results printed to stdout during code execution of that block. Note how this header argument syntax is different from the function call syntax.

‘<end header arguments>’
End header arguments affect the results returned by the code block. For example, ‘:results html’ wraps the results in a ‘#+BEGIN_EXPORT html’ block before inserting the results in the Org buffer.

Limit code block evaluation

The ‘eval’ header argument can limit evaluation of specific code blocks and ‘CALL’ keyword. It is useful for protection against evaluating untrusted code blocks by prompting for a confirmation.

‘yes’
Org evaluates the source code, possibly asking permission according to `org-confirm-babel-evaluate`.

‘never’ or ‘no’
Org never evaluates the source code.

‘query’
Org prompts the user for permission to evaluate the source code.

‘never-export’ or ‘no-export’
Org does not evaluate the source code when exporting, yet the user can evaluate it interactively.

‘query-export’

Org prompts the user for permission to evaluate the source code during export.

If ‘eval’ header argument is not set, then Org determines whether to evaluate the source code from the `org-confirm-babel-evaluate` variable (see Code Evaluation and Security Issues).

Cache results of evaluation

The ‘cache’ header argument is for caching results of evaluating code blocks. Caching results can avoid re-evaluating a code block that have not changed since the previous run. To benefit from the cache and avoid redundant evaluations, the source block must have a result already present in the buffer, and neither the header arguments—including the value of ‘var’ references—nor the text of the block itself has changed since the result was last computed. This feature greatly helps avoid long-running calculations. For some edge cases, however, the cached results may not be reliable.

The caching feature is best for when code blocks are pure functions, that is functions that return the same value for the same input arguments (see Environment of a Code Block), and that do not have side effects, and do not rely on external variables other than the input arguments. Functions that depend on a timer, file system objects, and random number generators are clearly unsuitable for caching.

A note of warning: when ‘cache’ is used in a session, caching may cause unexpected results.

When the caching mechanism tests for any source code changes, it does not expand noweb style references (see Noweb Reference Syntax).

The ‘cache’ header argument can have one of two values: ‘yes’ or ‘no’.

‘no’
Default. No caching of results; code block evaluated every time.

‘yes’
Whether to run the code or return the cached results is determined by comparing the SHA1 hash value of the combined code block and arguments passed to it. This hash value is packed on the ‘#+RESULTS:’ line from previous evaluation. When hash values match, Org does not evaluate the code block. When hash values mismatch, Org evaluates the code block, inserts the results, recalculates the hash value, and updates ‘#+RESULTS:’ line.

In this example, both functions are cached. But ‘caller’ runs only if the result from ‘random’ has changed since the last run.

```
#+NAME: random
#+BEGIN_SRC R :cache yes
  runif(+1)
#+END_SRC

#+RESULTS[a2a72cd647ad44515fab62e144796432793d68e1]: random
0.4659510825295

#+NAME: caller
#+BEGIN_SRC emacs-lisp :var x=random :cache yes
  x
#+END_SRC

#+RESULTS[bec9c8724e397d5df3b696502df3ed7892fc4f5f]: caller
0.254227238707244
```

Next: Exporting Code Blocks, Previous: Evaluating Code Blocks, Up: Working with Source Code [Contents][Index]

16.6 Results of Evaluation

How Org handles results of a code block execution depends on many header arguments working together. The primary determinant, however, is the ‘`results`’ header argument. It accepts four classes of options. Each code block can take only one option per class:

Collection
For how the results should be collected from the code block;

Type
For which type of result the code block will return; affects how Org processes and inserts results in the Org buffer;

Format
For the result; affects how Org processes results;

Handling
For inserting results once they are properly formatted.

Collection

Collection options specify the results. Choose one of the options; they are mutually exclusive.

‘`value`’
Default for most Babel libraries[153]. Functional mode. Org gets the value by wrapping the code in a function definition in the language of the source block. That is why when using ‘`:results value`’, code should execute like a function and return a value. For languages like Python, an explicit `return` statement is mandatory when using ‘`:results value`’. Result is the value returned by the last statement in the code block.

When evaluating the code block in a session (see Environment of a Code Block), Org passes the code to an interpreter running as an interactive Emacs inferior process. Org gets the value from the source code interpreter’s last statement output. Org has to use language-specific

methods to obtain the value. For example, from the variable `_` in Ruby, and the value of `.Last.value` in R.

‘`output`’
Scripting mode. Org passes the code to an external process running the interpreter. Org returns the contents of the standard output stream as text results.

When using a session, Org passes the code to the interpreter running as an interactive Emacs inferior process. Org concatenates any text output from the interpreter and returns the collection as a result.

Type

Type tells what result types to expect from the execution of the code block. Choose one of the options; they are mutually exclusive.

The default behavior is to automatically determine the result type. The result type detection depends on the code block language, as described in the documentation for individual languages. See Languages.

‘`table`’
‘`vector`’
Interpret the results as an Org table. If the result is a single value, create a table with one row and one column. Usage example: ‘`:results value table`’.

In-between each table row or below the table headings, sometimes results have horizontal lines, which are also known as “hlines”. The ‘`hlines`’ argument with the default ‘`no`’ value strips such lines from the input table. For most code, this is desirable, or else those ‘`hline`’ symbols raise unbound variable errors. A ‘`yes`’ accepts such lines, as demonstrated in the following example.

```
#+NAME: many-cols
| a | b | c |
|---+---+---|
| d | e | f |
|---+---+---|
| g | h | i |

#+NAME: no-hline
#+BEGIN_SRC python :var tab=many-cols :hlines no
return tab
#+END_SRC

#+RESULTS: no-hline
| a | b | c |
| d | e | f |
| g | h | i |

#+NAME: hlines
#+BEGIN_SRC python :var tab=many-cols :hlines yes
return tab
#+END_SRC

#+RESULTS: hlines
| a | b | c |
|---+---+---|
| d | e | f |
|---+---+---|
| g | h | i |
```

‘`list`’
Interpret the results as an Org list. If the result is a single value, create a list of one element.

‘`scalar`’
‘`verbatim`’
Interpret literally and insert as quoted text. Do not create a table. Usage example: ‘`:results value verbatim`’.

‘`file`’
Interpret as a filename. Save the results of execution of the code block to that file, then insert a link to it. You can control both the filename and the description associated to the link.

Org first tries to generate the filename from the value of the ‘`file`’ header argument and the directory specified using the ‘`output-dir`’ header arguments. If ‘`output-dir`’ is not specified, Org assumes it is the current directory.

```
#+BEGIN_SRC asymptote :results value file :file circle.pdf :output-dir img/
  size(2cm);
  draw(unitcircle);
#+END_SRC
```

If ‘`file`’ header argument is missing, Org generates the base name of the output file from the name of the code block, and its extension from the ‘`file-ext`’ header argument. In that case, both the name and the extension are mandatory.

Result can also be interpreted as path to file. See ‘`:results link`’.

```
#+name: circle
#+BEGIN_SRC asymptote :results value file :file-ext pdf
  size(2cm);
  draw(unitcircle);
#+END_SRC
```

The ‘file-desc’ header argument defines the description (see Link Format) for the link. If ‘file-desc’ is present but has no value, the ‘file’ value is used as the link description. When this argument is not present, the description is omitted. If you want to provide the ‘file-desc’ argument but omit the description, you can provide it with an empty vector (i.e., :file-desc []).

By default, Org assumes that a table written to a file has TAB-delimited output. You can choose a different separator with the ‘sep’ header argument.

The ‘file-mode’ header argument defines the file permissions. To make it executable, use ‘:file-mode (identity #o755)’.

```
#+BEGIN_SRC shell :results file :file script.sh :file-mode (identity #o755)
  echo "#!/bin/bash"
  echo "echo Hello World"
#+END_SRC
```

Format

Format pertains to the type of the result returned by the code block. Choose one of the options; they are mutually exclusive. The default follows from the type specified above.

‘raw’
Interpreted as raw Org mode. Inserted directly into the buffer. Aligned if it is a table. Usage example: ‘:results value raw’.

‘code’
Result enclosed in a code block. Useful for parsing. Usage example: ‘:results value code’.

‘drawer’
Results are added directly to the Org file as with ‘raw’, but are wrapped in a ‘RESULTS’ drawer or results macro (for inline code blocks), for later scripting and automated processing. Usage example: ‘:results value drawer’.

‘html’
Results enclosed in a ‘BEGIN_EXPORT html’ block. Usage example: ‘:results value html’.

‘latex’
Results enclosed in a ‘BEGIN_EXPORT latex’ block. Usage example: ‘:results value latex’.

‘link’
‘graphics’

When used along with ‘`file`’ type, the result is a link to the file specified in ‘`:file`’ header argument. However, unlike plain ‘`file`’ type, code block output is not written to the disk. The block is expected to generate the file by its side-effects only, as in the following example:

```
#+begin_src shell :results file link :file "org-mode-unicorn.svg"
  wget -c "https://orgmode.org/resources/img/org-mode-unicorn.svg"
#+end_src

#+RESULTS:
[[file:org-mode-unicorn.svg]]
```

If ‘`:file`’ header argument is omitted, interpret source block result as the file path.

‘`org`’
Results enclosed in a ‘`BEGIN_SRC org`’ block. For comma-escape, either TAB in the block, or export the file. Usage example: ‘`:results value org`’.

‘`pp`’
Result converted to pretty-print source code. Enclosed in a code block. Languages supported: Emacs Lisp, Python, and Ruby. Usage example: ‘`:results value pp`’.

The ‘`wrap`’ header argument unconditionally marks the results block by appending strings to ‘`#+BEGIN_`’ and ‘`#+END_`’. If no string is specified, Org wraps the results in a ‘`#+BEGIN_results`’ … ‘`#+END_results`’ block. It takes precedent over the ‘`results`’ value listed above. E.g.,

```
#+BEGIN_SRC emacs-lisp :results html :wrap EXPORT markdown
"<blink>Welcome back to the 90's</blink>"
#+END_SRC

#+RESULTS:
#+BEGIN_EXPORT markdown
<blink>Welcome back to the 90's</blink>
#+END_EXPORT
```

Handling

Handling options after collecting the results. Choose one of the options; they are mutually exclusive.

‘`replace`’
Default. Insert results in the Org buffer. Remove previous results. Usage example: ‘`:results output replace`’.

‘`silent`’
Do not insert results in the Org mode buffer, but echo them in the minibuffer. Usage example: ‘`:results output silent`’.

‘none’
Compute results, but do not do anything with them. No inserting in the Org mode buffer nor echo them in the minibuffer. The results can still be used when referenced from another code block. Usage example: ‘`:results none`’.

‘discard’
Ignore the results completely. This option is similar to ‘none’, but no processing is performed on the return value. Calling the code block programmatically (see How to evaluate source code) or by reference (see Passing arguments and Noweb Reference Syntax) will always yield `nil`.

‘append’
Append results to the Org buffer. Latest results are at the bottom. Does not remove previous results. Usage example: ‘`:results output append`’.

‘prepend’
Prepend results to the Org buffer. Latest results are at the top. Does not remove previous results. Usage example: ‘`:results output prepend`’.

Post-processing

The ‘`post`’ header argument is for post-processing results from block evaluation. When ‘`post`’ has any value, Org binds the results to `*this*` variable for easy passing to ‘`var`’ header argument specifications (see Environment of a Code Block). That makes results available to other code blocks, or even for direct Emacs Lisp code execution.

The following two examples illustrate ‘`post`’ header argument in action. The first one shows how to attach an ‘ATTR_LATEX’ keyword using ‘`post`’.

```
#+NAME: attr_wrap
#+BEGIN_SRC sh :var data="" :var width="\\textwidth" :results output
  echo "#+ATTR_LATEX: :width $width"
  echo "$data"
#+END_SRC

#+HEADER: :file /tmp/it.png
#+BEGIN_SRC dot :post attr_wrap(width="5cm", data=*this*) :results drawer
  digraph{
          a -> b;
          b -> c;
          c -> a;
  }
#+end_src

#+RESULTS:
:RESULTS:
#+ATTR_LATEX :width 5cm
[[file:/tmp/it.png]]
:END:
```

The second example shows use of ‘`colnames`’ header argument in ‘`post`’ to pass data between code blocks.

```
#+NAME: round-tbl
#+BEGIN_SRC emacs-lisp :var tbl="" fmt="%.3f"
  (mapcar (lambda (row)
            (mapcar (lambda (cell)
                      (if (numberp cell)
                          (format fmt cell)
                        cell))
                    row))
          tbl)
#+end_src

#+BEGIN_SRC R :colnames yes :post round-tbl[:colnames yes](*this*)
  set.seed(42)
  data.frame(foo=rnorm(1))
#+END_SRC

#+RESULTS:
|   foo |
|-------|
| 1.371 |
```

Next: Extracting Source Code, Previous: Results of Evaluation, Up: Working with Source Code [Contents][Index]

16.7 Exporting Code Blocks

It is possible to export the *code* of code blocks, the *results* of code block evaluation, *both* the code and the results of code block evaluation, or *none*. Org defaults to exporting *code* for most languages and *results* for inline code blocks. For some languages, such as ditaa, Org defaults to *results* both in ordinary source blocks and in inline source blocks. To export just the body of code blocks, see Literal Examples. To selectively export subtrees of an Org document, see Exporting.

The ‘`exports`’ header argument is to specify if that part of the Org file is exported to, say, HTML or LaTeX formats.

‘`code`’
The default. The body of code is included into the exported file. Example: ‘`:exports code`’.

‘`results`’
The results of evaluation of the code is included in the exported file. Example: ‘`:exports results`’.

‘`both`’
Both the code and results of evaluation are included in the exported file. Example: ‘`:exports both`’.

‘none’
Neither the code nor the results of evaluation is included in the exported file. Whether the code is evaluated at all depends on other options. Example: ‘`:exports none`’.

If a source block is named using ‘NAME’ keyword, the same name will be assigned to the results of evaluation. This way, fuzzy links pointing to the named source blocks exported using ‘`:exports results`’ will remain valid and point to the results of evaluation.

Results of evaluation of a named block can also be explicitly named using a separate ‘NAME’ keyword. The name value set via ‘NAME’ keyword will be preferred over the parent source block.

```
#+NAME: code name
#+BEGIN_SRC emacs-lisp :exports both value
(+ 1 2)
#+END_SRC

#+NAME: results name
#+RESULTS: code name
3

This [[code name][link]] will point to the code block.
Another [[results name][link]] will point to the results.
```

Explicit setting of the result name may be necessary when a named code block is exported using ‘`:exports both`’. Links to such block may arbitrarily point either to the code block or to its results when results do not have a distinct name.

Note that all the links pointing to a source block exported using ‘`:exports none`’ will be broken. This will make export process fail, unless broken links are allowed during export (see Export Settings).

To stop Org from evaluating code blocks to speed exports, use the header argument ‘`:eval never-export`’ (see Evaluating Code Blocks). To stop Org from evaluating code blocks for greater security, set the `org-export-use-babel` variable to `nil`, but understand that header arguments will have no effect.

Turning off evaluation comes in handy when batch processing. For example, markup languages for wikis, which have a high risk of untrusted code. Stopping code block evaluation also stops evaluation of all header arguments of the code block. This may not be desirable in some circumstances. So during export, to allow evaluation of just the header arguments but not any code evaluation in the source block, set ‘`:eval never-export`’ (see Evaluating Code Blocks).

Org never evaluates code blocks in commented subtrees when exporting (see Comment Lines). On the other hand, Org does evaluate code blocks in subtrees excluded from export (see Export Settings).

Next: Languages, Previous: Exporting Code Blocks, Up: Working with Source Code [Contents] [Index]

16.8 Extracting Source Code

Extracting source code from code blocks is a basic task in literate programming. Org has features to make this easy. In literate programming parlance, documents on creation are *woven* with code and documentation, and on export, the code is tangled for execution by a computer.

Org facilitates weaving and tangling for producing, maintaining, sharing, and exporting literate programming documents. Org provides extensive customization options for extracting source code.

When Org tangles code blocks, it expands, merges, and transforms them. Then Org recomposes them into one or more separate files, as configured through the options. During this tangling process, Org expands variables in the source code, and resolves any noweb style references (see Noweb Reference Syntax).

Header arguments

The ‘`tangle`’ header argument specifies if the code block is exported to source file(s).

‘`yes`’
Export the code block to source file. The file name for the source file is derived from the name of the Org file, and the file extension is derived from the source code language identifier. Example: ‘`:tangle yes`’.

‘`no`’
The default. Do not extract the code in a source code file. Example: ‘`:tangle no`’.

FILENAME
Export the code block to source file whose file name is derived from any string passed to the ‘`tangle`’ header argument. Org derives the file name as being relative to the directory of the Org file’s location. Example: ‘`:tangle FILENAME`’.

The ‘`mkdirp`’ header argument creates parent directories for tangled files if the directory does not exist. A ‘`yes`’ value enables directory creation whereas ‘`no`’ inhibits it.

The ‘`comments`’ header argument controls inserting comments into tangled files. These are above and beyond whatever comments may already exist in the code block.

‘`no`’
The default. Do not insert any extra comments during tangling.

‘`link`’
Wrap the code block in comments. Include links pointing back to the place in the Org file from where the code was tangled.

‘`yes`’
Kept for backward compatibility; same as ‘`link`’.

‘`org`’
Nearest headline text from Org file is inserted as comment. The exact text that is inserted is picked from the leading context of the source block.

‘`both`’
Includes both ‘`link`’ and ‘`org`’ options.

‘`noweb`’

Includes ‘link’ option, expands noweb references (see Noweb Reference Syntax), and wraps them in link comments inside the body of the code block.

The ‘padline’ header argument controls insertion of newlines to pad source code in the tangled file.

‘yes’
Default. Insert a newline before and after each code block in the tangled file.

‘no’
Do not insert newlines to pad the tangled code blocks.

The ‘shebang’ header argument can turn results into executable script files. By setting it to a string value—for example, ‘`:shebang "#!/bin/bash"`’—Org inserts that string as the first line of the tangled file that the code block is extracted to. Org then turns on the tangled file’s executable permission.

The ‘tangle-mode’ header argument specifies what permissions to set for tangled files by `set-file-modes`. Permissions are given by an octal value, which can be provided calling the `identity` function on an elisp octal value. For instance, to create a read-only file one may use ‘`:tangle-mode (identity #o444)`’. To reduce the verbosity required, a octal shorthand is defined, ‘oXXX’ (‘o’ for octal). Using this, our read-only example is ‘`:tangle-mode o444`’. Omitting the ‘o’ prefix will cause the argument to be interpreted as an integer, which can lead to unexpected results (‘444’ is the same as ‘o674’). Two other shorthands are recognized, ls-style strings like ‘rw-r--r--’, and chmod-style permissions like ‘g+w’. Note that chmod-style permissions are based on `org-babel-tangle-default-file-mode`, which is ‘#o544’ by default.

When ‘`:tangle-mode`’ and ‘`:shebang`’ are both specified, the give ‘`:tangle-mode`’ will override the permissions from ‘`:shebang`’. When multiple source code blocks tangle to a single file with conflicting ‘`:tangle-mode`’ header arguments, Org’s behavior is undefined.

By default Org expands code blocks during tangling. The ‘no-expand’ header argument turns off such expansions. Note that one side-effect of expansion by `org-babel-expand-src-block` also assigns values (see Environment of a Code Block) to variables. Expansions also replace noweb references with their targets (see Noweb Reference Syntax). Some of these expansions may cause premature assignment, hence this option. This option makes a difference only for tangling. It has no effect when exporting since code blocks for execution have to be expanded anyway.

Functions

`org-babel-tangle` ¶
Tangle the current file. Bound to `C-c C-v t`.

With prefix argument only tangle the current code block.

`org-babel-tangle-file` ¶
Choose a file to tangle. Bound to `C-c C-v f`.

Tangle hooks

`org-babel-pre-tangle-hook` ¶
This hook is run before the tangle process begins. The active buffer is buffer to be tangled.

`org-babel-tangle-body-hook` ¶
This hook is run from a temporary buffer containing expanded code of every tangled code block. The hook can modify the expanded code as needed. The contents of the current buffer will be used as actual code block expansion.

`org-babel-post-tangle-hook` ¶
This hook is run from within code files tangled by `org-babel-tangle`, making it suitable for post-processing, compilation, and evaluation of code in the tangled files.

`org-babel-tangle-finished-hook` ¶
This hook is run after post-tangle hooks, in the original buffer.

Jumping between code and Org

Debuggers normally link errors and messages back to the source code. But for tangled files, we want to link back to the Org file, not to the tangled source file. To make this extra jump, Org uses `org-babel-tangle-jump-to-org` function with two additional source code block header arguments:

1. Set '`padline`' to true—this is the default setting.
2. Set '`comments`' to '`link`', which makes Org insert links to the Org file.

16.9 Languages

Code blocks in dozens of languages are supported. See Worg website for language specific documentation.

By default, only Emacs Lisp is enabled for evaluation. To enable or disable other languages, customize the `org-babel-load-languages` variable either through the Emacs customization interface, or by adding code to the init file as shown next.

In this example, evaluation is disabled for Emacs Lisp, and enabled for R.

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((emacs-lisp . nil)
   (R . t)))
```

Note that this is not the only way to enable a language. Org also enables languages when loaded with `require` statement. For example, the following enables execution of Clojure code blocks:

```
(require 'ob-clojure)
```

16.10 Editing Source Code

Use `C-c '` to edit the current code block. It opens a new major mode edit buffer containing the body of the source code block, ready for any edits. Use `C-c '` again to close the buffer and return to the Org buffer.

`C-x C-s` saves the buffer and updates the contents of the Org buffer. Set `org-edit-src-auto-save-idle-delay` to save the base buffer after a certain idle delay time. Set `org-edit-src-turn-on-auto-save` to auto-save this buffer into a separate file using Auto-save mode.

While editing the source code in the major mode, the Org Src minor mode remains active. It provides these customization variables as described below. For even more variables, look in the customization group `org-edit-structure`.

`org-src-lang-modes` ¶
If an Emacs major-mode named `<LANG>-mode` exists, where *<LANG>* is the language identifier from code block's header line, then the edit buffer uses that major mode. Use this variable to arbitrarily map language identifiers to major modes.

When language identifier is omitted in the src block, Org mode's behavior is undefined.

`org-src-window-setup` ¶
For specifying Emacs window arrangement when the new edit buffer is created.

`org-src-preserve-indentation` ¶
Default is `nil`. Source code is indented. This indentation applies during export or tangling, and depending on the context, may alter leading spaces and tabs. When non-`nil`, source code is aligned with the leftmost column. No lines are modified during export or tangling, which is very useful for white-space sensitive languages, such as Python.

`org-src-ask-before-returning-to-edit-buffer` ¶
When `nil`, Org returns to the edit buffer without further prompts. The default prompts for a confirmation.

Fontification of code blocks can give visual separation of text and code on the display page. Set `org-src-fontify-natively` to non-`nil` to turn on native code fontification in the *Org* buffer. The fontification follows the major mode used to edit the code block (see `org-src-lang-modes` above).

To further customize the appearance of `org-block` for specific languages, customize `org-src-block-faces`. The following example shades the background of regular blocks, and colors source blocks only for Python and Emacs Lisp languages.

```
(require 'color)
(set-face-attribute 'org-block nil :background
                    (color-darken-name
                     (face-attribute 'default :background) 3))

(setq org-src-block-faces '(("emacs-lisp" (:background "#EEE2FF"))
                            ("python" (:background "#E5FFB8"))))
```

Next: Library of Babel, Previous: Editing Source Code, Up: Working with Source Code [Contents][Index]

16.11 Noweb Reference Syntax

Source code blocks can include references to other source code blocks, using a noweb[154] style syntax:

```
<<CODE-BLOCK-ID>>
```

where *CODE-BLOCK-ID* refers to either the ‘NAME’ of a single source code block, or a collection of one or more source code blocks sharing the same ‘`noweb-ref`’ header argument (see Using Header Arguments). Org can replace such references with the source code of the block or blocks being referenced, or, in the case of a single source code block named with ‘NAME’, with the results of an evaluation of that block.

The ‘noweb’ header argument controls expansion of noweb syntax references. Expansions occur when source code blocks are evaluated, tangled, or exported.

‘no’
Default. No expansion of noweb syntax references in the body of the code when evaluating, tangling, or exporting.

‘`yes`’
Expansion of noweb syntax references in the body of the code block when evaluating, tangling, or exporting.

‘`tangle`’
Expansion of noweb syntax references in the body of the code block when tangling. No expansion when evaluating or exporting.

‘`strip-tangle`’
Expansion of noweb syntax references in the body of the code block when evaluating or exporting. Removes noweb syntax references when exporting.

‘`no-export`’
Expansion of noweb syntax references in the body of the code block when evaluating or tangling. No expansion when exporting.

‘`strip-export`’
Expansion of noweb syntax references in the body of the code block when expanding prior to evaluating or tangling. Removes noweb syntax references when exporting.

‘`eval`’
Expansion of noweb syntax references in the body of the code block only before evaluating.

In the most simple case, the contents of a single source block is inserted within other blocks. Thus, in following example,

```
#+NAME: initialization
#+BEGIN_SRC emacs-lisp
  (setq sentence "Never a foot too far, even.")
#+END_SRC

#+BEGIN_SRC emacs-lisp :noweb yes
  <<initialization>>
  (reverse sentence)
#+END_SRC
```

the second code block is expanded as

```
#+BEGIN_SRC emacs-lisp :noweb yes
  (setq sentence "Never a foot too far, even.")
  (reverse sentence)
#+END_SRC
```

Note that noweb expansion does not automatically carry over '`:var`' header arguments[155].

You may also include the contents of multiple blocks sharing a common '`noweb-ref`' header argument, which can be set at the file, subtree, or code block level. In the example Org file shown next, the body of the source code in each block is extracted for concatenation to a pure code file when tangled.

```
#+BEGIN_SRC sh :tangle yes :noweb yes :shebang #!/bin/sh
  <<fullest-disk>>
#+END_SRC
* the mount point of the fullest disk
  :PROPERTIES:
  :header-args: :noweb-ref fullest-disk
  :END:

** query all mounted disks
#+BEGIN_SRC sh
  df \
#+END_SRC

** strip the header row
#+BEGIN_SRC sh
  |sed '1d' \
#+END_SRC

** output mount point of fullest disk
#+BEGIN_SRC sh
  |awk '{if (u < +$5) {u = +$5; m = $6}} END {print m}'
#+END_SRC
```

By default a newline separates each noweb reference concatenation. To use a different separator, edit the '`noweb-sep`' header argument.

Alternatively, Org can include the results of evaluation of a single code block rather than its body [[156]. Evaluation occurs when parentheses, possibly including arguments, are appended to the code block name, as shown below.

```
<<NAME(optional arguments)>>
```

Note that in this case, a code block name set by '`NAME`' keyword is required; the reference set by '`noweb-ref`' will not work when evaluation is desired.

Here is an example that demonstrates how the exported content changes when noweb style references are used with parentheses versus without. Given:

```
#+NAME: some-code
#+BEGIN_SRC python :var num=0 :results output :exports none
print(num*10)
#+END_SRC
```

this code block:

```
#+BEGIN_SRC text :noweb yes
  <<some-code>>
#+END_SRC
```

expands to:

```
print(num*10)
```

Below, a similar noweb style reference is used, but with parentheses, while setting a variable ‘num’ to 10:

```
#+BEGIN_SRC text :noweb yes
  <<some-code(num=10)>>
#+END_SRC
```

Note that the expansion now contains the results of the code block ‘some-code’, not the code block itself:

```
100
```

Noweb insertions honor prefix characters that appear before the noweb syntax reference. This behavior is illustrated in the following example. Because the ‘<<example>>’ noweb reference appears behind the SQL comment syntax, each line of the expanded noweb reference is commented. With:

```
#+NAME: example
#+BEGIN_SRC text
  this is the
  multi-line body of example
#+END_SRC
```

this code block:

```
#+BEGIN_SRC sql :noweb yes
 ---<<example>>
#+END_SRC
```

expands to:

```
#+BEGIN_SRC sql :noweb yes
 ---this is the
 ---multi-line body of example
#+END_SRC
```

Since this change does not affect noweb replacement text without newlines in them, inline noweb references are acceptable.

This feature can also be used for management of indentation in exported code snippets. With:

```
#+NAME: if-true
#+BEGIN_SRC python :exports none
print('do things when true')
#+end_src

#+name: if-false
#+begin_src python :exports none
print('do things when false')
#+end_src
```

this code block:

```
#+begin_src python :noweb yes :results output
if true:
    <<if-true>>
else:
    <<if-false>>
#+end_src
```

expands to:

```
if true:
    print('do things when true')
else:
    print('do things when false')
```

This prefix behavior can be turned off in a block by setting the ‘`noweb-prefix`’ header argument to ‘no’, as in:

```
#+BEGIN_SRC elisp :noweb-prefix no
  (setq example-data "<<example>>")
#+END_SRC
```

which expands to:

```
(setq example-data "this is the
multi-line body of example")
```

When in doubt about the outcome of a source code block expansion, you can preview the results with the following command:

`C-c C-v v` or **`C-c C-v C-v`** (**`org-babel-expand-src-block`**)¶
Expand the current source code block according to its header arguments and pop open the results in a preview buffer.

16.12 Library of Babel

The “Library of Babel” is a collection of code blocks. Like a function library, these code blocks can be called from other Org files. A collection of useful code blocks is available on Worg. For remote code block evaluation syntax, see Evaluating Code Blocks.

For any user to add code to the library, first save the code in regular code blocks of an Org file, and then load the Org file with `org-babel-lob-ingest`, which is bound to `C-c C-v i`.

Next: Batch Execution, Previous: Library of Babel, Up: Working with Source Code [Contents] [Index]

16.13 Key bindings and Useful Functions

Many common Org mode key sequences are re-bound depending on the context.

Active key bindings in code blocks:

Key binding	Function
`C-c C-c`	`org-babel-execute-src-block`
`C-c C-o`	`org-babel-open-src-block-result`
`M-UP`	`org-babel-load-in-session`
`M-DOWN`	`org-babel-pop-to-session`

Active key bindings in Org mode buffer:

Key binding	Function
`C-c C-v p` or `C-c C-v C-p`	`org-babel-previous-src-block`
`C-c C-v n` or `C-c C-v C-n`	`org-babel-next-src-block`
`C-c C-v e` or `C-c C-v C-e`	`org-babel-execute-maybe`
`C-c C-v o` or `C-c C-v C-o`	`org-babel-open-src-block-result`
`C-c C-v v` or `C-c C-v C-v`	`org-babel-expand-src-block`
`C-c C-v u` or `C-c C-v C-u`	`org-babel-goto-src-block-head`
`C-c C-v g` or `C-c C-v C-g`	`org-babel-goto-named-src-block`
`C-c C-v r` or `C-c C-v C-r`	`org-babel-goto-named-result`
`C-c C-v b` or `C-c C-v C-b`	`org-babel-execute-buffer`
`C-c C-v s` or `C-c C-v C-s`	`org-babel-execute-subtree`
`C-c C-v d` or `C-c C-v C-d`	`org-babel-demarcate-block`
`C-c C-v t` or `C-c C-v C-t`	`org-babel-tangle`
`C-c C-v f` or `C-c C-v C-f`	`org-babel-tangle-file`
`C-c C-v c` or `C-c C-v C-c`	`org-babel-check-src-block`
`C-c C-v j` or `C-c C-v C-j`	`org-babel-insert-header-arg`
`C-c C-v l` or `C-c C-v C-l`	`org-babel-load-in-session`
`C-c C-v i` or `C-c C-v C-i`	`org-babel-lob-ingest`
`C-c C-v I` or `C-c C-v C-I`	`org-babel-view-src-block-info`
`C-c C-v z` or `C-c C-v C-z`	`org-babel-switch-to-session-with-code`
`C-c C-v a` or `C-c C-v C-a`	`org-babel-sha1-hash`

Key binding	Function
`C-c C-v h` or `C-c C-v C-h`	`org-babel-describe-bindings`
`C-c C-v x` or `C-c C-v C-x`	`org-babel-do-key-sequence-in-edit-buffer`

16.14 Batch Execution

Org mode features, including working with source code facilities can be invoked from the command line. This enables building shell scripts for batch processing, running automated system tasks, and expanding Org mode's usefulness.

The sample script shows batch processing of multiple files using `org-babel-tangle`.

```
#!/bin/sh
# Tangle files with Org mode
#
emacs -Q --batch --eval "
    (progn
      (require 'ob-tangle)
      (dolist (file command-line-args-left)
        (with-current-buffer (find-file-noselect file)
          (org-babel-tangle))))
  " "$@"
```

Next: Hacking, Previous: Working with Source Code, Up: The Org Manual [Contents][Index]

17 Miscellaneous

17.1 Completion

Org has in-buffer completions. Unlike minibuffer completions, which are useful for quick command interactions, Org's in-buffer completions are more suitable for content creation in Org documents. Type one or more letters and invoke the hot key to complete the text in-place. Depending on the context and the keys, Org offers different types of completions. No minibuffer is involved. Such mode-specific hot keys have become an integral part of Emacs and Org provides several shortcuts.

`M-TAB` ¶
Complete word at point.

- At the beginning of an empty headline, complete TODO keywords.
- After '\', complete TeX symbols supported by the exporter.
- After '`:`' in a headline, complete tags. Org deduces the list of tags from the 'TAGS' in-buffer option (see Setting Tags), the variable `org-tag-alist`, or from all tags used in the current buffer.
- After '`:`' and not in a headline, complete property keys. The list of keys is constructed dynamically from all keys used in the current buffer.
- After '`[[`', complete link abbreviations (see Link Abbreviations).
- After '`[[*`', complete headlines in the current buffer so that they can be used in search links like: '`[[*find this headline]]`'

- After '#+', complete the special keywords like 'TYP_TODO' or file-specific 'OPTIONS'. After option keyword is complete, pressing `M-TAB` again inserts example settings for this keyword.
- After 'STARTUP' keyword, complete startup items.
- When point is anywhere else, complete dictionary words using Ispell.

17.2 Structure Templates

With just a few keystrokes, it is possible to insert empty structural blocks, such as '#+BEGIN_SRC' … '#+END_SRC', or to wrap existing text in such a block.

`C-c C-,` (**org-insert-structure-template**) ¶
Prompt for a type of block structure, and insert the block at point. If the region is active, it is wrapped in the block. First prompts the user for keys, which are used to look up a structure type from the variable below. If the key is `TAB`, `RET`, or `SPC`, the user is prompted to enter a block type.

Available structure types are defined in `org-structure-template-alist`, see the docstring for adding or changing values.

Org Tempo expands snippets to structures defined in `org-structure-template-alist` and `org-tempo-keywords-alist`. For example, `< s TAB` creates a code block. Enable it by customizing `org-modules` or add '`(require 'org-tempo)`' to your Emacs init file[157].

a	'#+BEGIN_EXPORT ascii' … '#+END_EXPORT'
c	'#+BEGIN_CENTER' … '#+END_CENTER'
C	'#+BEGIN_COMMENT' … '#+END_COMMENT'
e	'#+BEGIN_EXAMPLE' … '#+END_EXAMPLE'
E	'#+BEGIN_EXPORT' … '#+END_EXPORT'
h	'#+BEGIN_EXPORT html' … '#+END_EXPORT'
l	'#+BEGIN_EXPORT latex' … '#+END_EXPORT'
q	'#+BEGIN_QUOTE' … '#+END_QUOTE'
s	'#+BEGIN_SRC' … '#+END_SRC'
v	'#+BEGIN_VERSE' … '#+END_VERSE'

17.3 Speed Keys

Single keystrokes can execute custom commands in an Org file when point is on a headline. Without the extra burden of a meta or modifier key, Speed Keys can speed navigation or execute custom commands. Besides faster navigation, Speed Keys may come in handy on small mobile devices that do not have full keyboards. Speed Keys may also work on TTY devices known for their problems when entering Emacs key chords.

By default, Org has Speed Keys disabled. To activate Speed Keys, set the variable `org-use-speed-commands` to a non-`nil` value. To trigger a Speed Key, point must be at the beginning of an Org headline, before any of the stars.

Org comes with a pre-defined list of Speed Keys. To add or modify Speed Keys, customize the option `org-speed-commands`. For more details, see the variable's docstring. With Speed Keys activated, `M-x org-speed-command-help`, or `?` when point is at the beginning of an Org headline, shows currently active Speed Keys, including the user-defined ones.

17.4 A Cleaner Outline View

Org's outline with stars and no indents can look cluttered for short documents. For *book-like* long documents, the effect is not as noticeable. Org provides an alternate stars and indentation scheme, as shown on the right in the following table. It displays only one star and indents text to line up with the heading:

```
* Top level headline             |    * Top level headline
** Second level                  |      * Second level
*** Third level                  |        * Third level
some text                        |          some text
*** Third level                  |        * Third level
more text                        |          more text
* Another top level headline     |    * Another top level headline
```

Org can achieve this in two ways, (1) by just displaying the buffer in this way without changing it, or (2) by actually indenting every line in the desired amount with hard spaces and hiding leading stars.

17.4.1 Org Indent Mode

To display the buffer in the indented view, activate Org Indent minor mode, using `M-x org-indent-mode`. Text lines that are not headlines are prefixed with virtual spaces to vertically align with the headline text[158].

To make more horizontal space, the headlines are shifted by two characters. Configure `org-indent-indentation-per-level` variable for a different number.

By default, Org Indent mode turns off `org-adapt-indentation` and does hide leading stars by locally setting `org-hide-leading-stars` to `t`: only one star on each headline is visible, the rest are masked with the same font color as the background. If you want to customize this default behavior, see `org-indent-mode-turns-on-hiding-stars` and `org-indent-mode-turns-off-org-adapt-indentation`.

To globally turn on Org Indent mode for all files, customize the variable `org-startup-indented`. To control it for individual files, use 'STARTUP' keyword as follows:

```
#+STARTUP: indent
#+STARTUP: noindent
```

17.4.2 Hard indentation

It is possible to use hard spaces to achieve the indentation instead, if the bare ASCII file should have the indented look also outside Emacs[159]. With Org's support, you have to indent all lines to line up with the outline headers. You would use these settings[160]:

```
(setq org-adapt-indentation t
      org-hide-leading-stars t
      org-odd-levels-only t)
```

Indentation of text below headlines (`org-adapt-indentation`) ¶
The first setting modifies paragraph filling, line wrapping, and structure editing commands to preserving or adapting the indentation as appropriate.

Hiding leading stars (`org-hide-leading-stars`) ¶
The second setting makes leading stars invisible by applying the face `org-hide` to them. For per-file preference, use these file 'STARTUP' options:

```
#+STARTUP: hidestars
#+STARTUP: showstars
```

Odd levels (`org-odd-levels-only`) ¶
The third setting makes Org use only odd levels, 1, 3, 5, …, in the outline to create more indentation. On a per-file level, control this with:

```
#+STARTUP: odd
#+STARTUP: oddeven
```

To convert a file between single and double stars layouts, use `M-x org-convert-to-odd-levels` and `M-x org-convert-to-oddeven-levels`.

Next: Dynamic Headline Numbering, Previous: A Cleaner Outline View, Up: Miscellaneous [Contents][Index]

17.5 Execute commands in the active region

When in an Org buffer and the region is active, some commands will apply to all the subtrees in the active region. For example, hitting `C-c C-s` when multiple headlines are within the active region will successively prompt you for a new schedule date and time. To disable this, set the option `org-loop-over-headlines-in-active-region` to non-`t`, activate the region and run the command normally.

`org-agenda-loop-over-headlines-in-active-region` is the equivalent option of the agenda buffer, where you can also use bulk editing of selected entries.

Not all commands can loop in the active region and what subtrees or headlines are considered can be refined: see the docstrings of these options for more details.

Next: The Very Busy `C-c C-c` Key, Previous: Execute commands in the active region, Up: Miscellaneous [Contents][Index]

17.6 Dynamic Headline Numbering

The Org Num minor mode, toggled with `M-x org-num-mode`, displays outline numbering on top of headlines. It also updates it automatically upon changes to the structure of the document.

By default, all headlines are numbered. You can limit numbering to specific headlines according to their level, tags, 'COMMENT' keyword, or 'UNNUMBERED' property. Set `org-num-max-level`, `org-num-skip-tags`, `org-num-skip-commented`, `org-num-skip-unnumbered`, or `org-num-skip-footnotes` accordingly.

If `org-num-skip-footnotes` is non-`nil`, footnotes sections (see Creating Footnotes) are not numbered either.

You can control how the numbering is displayed by setting `org-num-face` and `org-num-format-function`.

You can also turn this mode globally for all Org files by setting the option `org-startup-numerated` to 't', or locally on a file by using '`#+startup: num`'.

17.7 The Very Busy `C-c C-c` Key

The `C-c C-c` key in Org serves many purposes depending on the context. It is probably the most over-worked, multi-purpose key combination in Org. Its uses are well documented throughout this manual, but here is a consolidated list for easy reference.

- If column view (see Column View) is on, exit column view.
- If any highlights shown in the buffer from the creation of a sparse tree, or from clock display, remove such highlights.
- If point is in one of the special 'KEYWORD' lines, scan the buffer for these lines and update the information. Also reset the Org file cache used to temporary store the contents of URLs used as values for keywords like 'SETUPFILE'.
- If point is inside a table, realign the table.
- If point is on a 'TBLFM' keyword, re-apply the formulas to the entire table.
- If the current buffer is a capture buffer, close the note and file it. With a prefix argument, also jump to the target location after saving the note.
- If point is on a '<<<target>>>', update radio targets and corresponding links in this buffer.
- If point is on a property line or at the start or end of a property drawer, offer property commands.
- If point is at a footnote reference, go to the corresponding definition, and *vice versa*.
- If point is on a statistics cookie, update it.
- If point is in a plain list item with a checkbox, toggle the status of the checkbox.
- If point is on a numbered item in a plain list, renumber the ordered list.
- If point is on the '#+BEGIN' line of a dynamic block, the block is updated.
- If point is at a timestamp, fix the day name in the timestamp.

17.8 Summary of In-Buffer Settings

In-buffer settings start with '#+', followed by a keyword, a colon, one or more spaces, and then a word for each setting. Org accepts multiple settings on the same line. Org also accepts multiple lines for a keyword. This manual describes these settings throughout. A summary follows here.

`C-c C-c` activates any changes to the in-buffer settings. Closing and reopening the Org file in Emacs also activates the changes.

‘#+ARCHIVE: %s_done::’ ¶
Sets the archive location of the agenda file. The corresponding variable is `org-archive-location`.

‘#+CATEGORY’ ¶
Sets the category of the agenda file, which applies to the entire document.

‘#+COLUMNS: %25ITEM ...’ ¶
Set the default format for columns view. This format applies when columns view is invoked in locations where no ‘COLUMNS’ property applies.

‘#+CONSTANTS: name1=value1 ...’ ¶
Set file-local values for constants that table formulas can use. This line sets the local variable `org-table-formula-constants-local`. The global version of this variable is `org-table-formula-constants`.

‘#+FILETAGS: :tag1:tag2:tag3:’ ¶
Set tags that all entries in the file inherit from, including the top-level entries.

‘#+LINK: linkword replace’ ¶
Each line specifies one abbreviation for one link. Use multiple ‘LINK’ keywords for more, see Link Abbreviations. The corresponding variable is `org-link-abbrev-alist`.

‘#+PRIORITIES: highest lowest default’ ¶
This line sets the limits and the default for the priorities. All three must be either letters A–Z or numbers 0–9. The highest priority must have a lower ASCII number than the lowest priority.

‘#+PROPERTY: Property_Name Value’ ¶
This line sets a default inheritance value for entries in the current buffer, most useful for specifying the allowed values of a property.

‘#+SETUPFILE: file’ ¶
The setup file or a URL pointing to such file is for additional in-buffer settings. Org loads this file and parses it for any settings in it when Org opens the main file. If URL is specified, the contents are downloaded and stored in a temporary file cache. `C-c C-c` on the settings line re-parses and re-loads the file, and also resets the temporary file cache.

Org also parses and loads *in-buffer settings* from the setup file during normal exporting process. Org parses the *in-buffer settings* as if it was included in the containing Org buffer. The rest of the contents of setup file is ignored.

To visit the setup file—not a URL—use `C-c '` while point is on the line with the setup file name.

‘#+STARTUP:’ ¶
Startup options Org uses when first visiting a file.

The first set of options deals with the initial visibility of the outline tree. The corresponding variable for global default settings is `org-startup-folded` with a default value of `showeverything`.

‘overview’	Top-level headlines only.
‘content’	All headlines.
‘showall’	No folding on any entry.
‘show2levels’	Headline levels 1-2.
‘show3levels’	Headline levels 1-3.
‘show4levels’	Headline levels 1-4.
‘show5levels’	Headline levels 1-5.
‘showeverything’	Show even drawer contents.

Dynamic virtual indentation is controlled by the variable `org-startup-indented`[161].

‘indent’	Start with Org Indent mode turned on.
‘noindent’	Start with Org Indent mode turned off.

Dynamic virtual numeration of headlines is controlled by the variable `org-startup-numerated`.

‘num’	Start with Org num mode turned on.
‘nonum’	Start with Org num mode turned off.

Aligns tables consistently upon visiting a file. The corresponding variable is `org-startup-align-all-tables` with `nil` as default value.

‘align’	Align all tables.
‘noalign’	Do not align tables on startup.

Shrink table columns with a width cookie. The corresponding variable is `org-startup-shrink-all-tables` with `nil` as default value.

When visiting a file, inline images can be automatically displayed. The corresponding variable is `org-startup-with-inline-images`, with a default value `nil` to avoid delays when visiting a file.

‘inlineimages’	Show inline images.
‘noinlineimages’	Do not show inline images on startup.

Bracket links in Org buffers are displayed hiding the link path and brackets. For example, ‘`[[https://orgmode.org][Org Website]]`’ is, by default, displayed as “Org Website”, hiding the link itself and just displaying its description. Alternatively, the links can be displayed in full. The corresponding variable is `org-link-descriptive`.

‘descriptivelinks’	Hide path and brackets in links.

‘literallinks’	Do not hide anything.

Logging the closing and reopening of TODO items and clock intervals can be configured using these options (see variables `org-log-done`, `org-log-note-clock-out`, and `org-log-repeat`).

‘logdone’	Record a timestamp when an item is marked as done.
‘lognotedone’	Record timestamp and a note when DONE.
‘nologdone’	Do not record when items are marked as done.
‘logrepeat’	Record a time when reinstating a repeating item.
‘lognoterepeat’	Record a note when reinstating a repeating item.
‘nologrepeat’	Do not record when reinstating repeating item.
‘lognoteclock-out’	Record a note when clocking out.
‘nologlognoteclock-out’	Do not record a note when clocking out.
‘logreschedule’	Record a timestamp when scheduling time changes.
‘lognotereschedule’	Record a note when scheduling time changes.
‘nologreschedule’	Do not record when a scheduling date changes.
‘logredeadline’	Record a timestamp when deadline changes.
‘lognoteredeadline’	Record a note when deadline changes.
‘nologredeadline’	Do not record when a deadline date changes.
‘logrefile’	Record a timestamp when refiling.
‘lognoterefile’	Record a note when refiling.
‘nologrefile’	Do not record when refiling.

Here are the options for hiding leading stars in outline headings, and for indenting outlines. The corresponding variables are `org-hide-leading-stars` and `org-odd-levels-only`, both with a default setting `nil` (meaning ‘showstars’ and ‘oddeven’).

‘hidestars’	Make all but one of the stars starting a headline invisible.
‘showstars’	Show all stars starting a headline.
‘indent’	Virtual indentation according to outline level.
‘noindent’	No virtual indentation according to outline level.
‘odd’	Allow only odd outline levels (1, 3, …).
‘oddeven’	Allow all outline levels.

To turn on custom format overlays over timestamps (variables `org-display-custom-times` and `org-timestamp-custom-formats`), use:

‘`customtime`’	Overlay custom time format.

The following options influence the table spreadsheet (variable `constants-unit-system`).

‘`constcgs`’	‘`constants.el`’ should use the c-g-s unit system.
‘`constSI`’	‘`constants.el`’ should use the SI unit system.

To influence footnote settings, use the following keywords. The corresponding variables are `org-footnote-define-inline`, `org-footnote-auto-label`, and `org-footnote-auto-adjust`.

‘`fninline`’	Define footnotes inline.
‘`fnnoinline`’	Define footnotes in separate section.
‘`fnlocal`’	Define footnotes near first reference, but not inline.
‘`fnprompt`’	Prompt for footnote labels.
‘`fnauto`’	Create ‘`[fn:1]`’-like labels automatically (default).
‘`fnconfirm`’	Offer automatic label for editing or confirmation.
‘`fnadjust`’	Automatically renumber and sort footnotes.
‘`nofnadjust`’	Do not renumber and sort automatically.
‘`fnanon`’	Create anonymous footnotes with `org-footnote-new`.

To hide blocks or drawers on startup, use these keywords. The corresponding variables are `org-hide-block-startup` and `org-hide-drawer-startup`.

‘`hideblocks`’	Hide all begin/end blocks on startup.
‘`nohideblocks`’	Do not hide blocks on startup.
‘`hidedrawers`’	Hide all begin/end blocks on startup.
‘`nohidedrawers`’	Do not hide blocks on startup.

The display of entities as UTF-8 characters is governed by the variable `org-pretty-entities` and the keywords

‘`entitiespretty`’	Show entities as UTF-8 characters where possible.
‘`entitiesplain`’	Leave entities plain.

‘`#+TAGS: TAG1(c1) TAG2(c2)`’¶

These lines (several such lines are allowed) specify the valid tags in this file, and (potentially) the corresponding *fast tag selection* keys. The corresponding variable is `org-tag-alist`.

‘#+TODO:’, ‘#+SEQ_TODO:’, ‘#+TYP_TODO:’ ¶
These lines set the TODO keywords and their interpretation in the current file. The corresponding variable is `org-todo-keywords`.

17.9 Regular Expressions

Org, as an Emacs mode, makes use of Elisp regular expressions for searching, matching and filtering. Elisp regular expressions have a somewhat different syntax then some common standards. Most notably, alternation is indicated using ‘`\|`’ and matching groups are denoted by ‘`\(...\)`’. For example the string ‘`home\|work`’ matches either ‘`home`’ or ‘`work`’.

For more information, see (emacs)Regular Expressions in Emacs.

17.10 Org Syntax

A reference document providing a formal description of Org’s syntax is available as a draft on Worg, initially written by Nicolas Goaziou. It defines Org’s core internal concepts such as “headlines”, “sections”, “affiliated keywords”, “(greater) elements” and “objects”. Each part of an Org document belongs to one of the previous categories.

To explore the abstract structure of an Org buffer, run this in a buffer:

```
M-: (org-element-parse-buffer) <RET>
```

It outputs a list containing the buffer’s content represented as an abstract structure. The export engine relies on the information stored in this list. Most interactive commands—e.g., for structure editing—also rely on the syntactic meaning of the surrounding context.

You can probe the syntax of your documents with the command

```
M-x org-lint <RET>
```

It runs a number of checks to find common mistakes. It then displays their location in a dedicated buffer, along with a description and a “trust level”, since false-positive are possible. From there, you can operate on the reports with the following keys:

`C-j`, `TAB`	Display the offending line
`RET`	Move point to the offending line
`g`	Check the document again
`h`	Hide all reports from the same checker
`i`	Also remove them from all subsequent checks
`S`	Sort reports by the column at point

17.11 Context Dependent Documentation

`C-c C-x I` in an Org file tries to open a suitable section of the Org manual depending on the syntax at point. For example, using it on a headline displays “Document Structure” section.

`q` closes the Info window.

17.12 Escape Character

You may sometimes want to write text that looks like Org syntax, but should really read as plain text. Org may use a specific escape character in some situations, i.e., a backslash in macros (see Macro Replacement) and links (see Link Format), or a comma in source and example blocks (see Literal Examples). In the general case, however, we suggest to use the zero width space. You can insert one with any of the following:

```
C-x 8 <RET> zero width space <RET>
C-x 8 <RET> 200B <RET>
```

For example, in order to write '`[[1,2]]`' as-is in your document, you may write instead

```
[X[1,2]]
```

where 'X' denotes the zero width space character.

17.13 Code Evaluation and Security Issues

Unlike plain text, running code comes with risk. Each source code block, in terms of risk, is equivalent to an executable file. Org therefore puts a few confirmation prompts by default. This is to alert the casual user from accidentally running untrusted code.

For users who do not run code blocks or write code regularly, Org's default settings should suffice. However, some users may want to tweak the prompts for fewer interruptions. To weigh the risks of automatic execution of code blocks, here are some details about code evaluation.

Org evaluates code in the following circumstances:

Source code blocks
Org evaluates source code blocks in an Org file during export. Org also evaluates a source code block with the `C-c C-c` key chord. Users exporting or running code blocks must load files only from trusted sources. Be wary of customizing variables that remove or alter default security measures.

User Option: org-confirm-babel-evaluate ¶
When `t`, Org prompts the user for confirmation before executing each code block. When `nil`, Org executes code blocks without prompting the user for confirmation. When this option is set to a custom function, Org invokes the function with these two arguments: the source code language and the body of the code block. The custom function must return either a `t` or `nil`, which determines if the user is prompted. Each source code language can be handled separately through this function argument.

For example, here is how to execute ditaa code blocks without prompting:

```
(defun my-org-confirm-babel-evaluate (lang body)
  (not (string= lang "ditaa")))  ;don't ask for ditaa
(setq org-confirm-babel-evaluate #'my-org-confirm-babel-evaluate)
```

Following 'shell' and 'elisp' links

Org has two link types that can directly evaluate code (see External Links). Because such code is not visible, these links have a potential risk. Org therefore prompts the user when it encounters such links. The customization variables are:

User Option: org-link-shell-confirm-function ¶
Function that prompts the user before executing a shell link.

User Option: org-link-elisp-confirm-function ¶
Function that prompts the user before executing an Emacs Lisp link.

Formulas in tables
Formulas in tables (see The Spreadsheet) are code that is evaluated either by the Calc interpreter, or by the Emacs Lisp interpreter.

17.14 Interaction with Other Packages

Org's compatibility and the level of interaction with other Emacs packages are documented here.

17.14.1 Packages that Org cooperates with

'`calc.el`' by Dave Gillespie ¶
Org uses the Calc package for implementing spreadsheet functionality in its tables (see The Spreadsheet). Org also uses Calc for embedded calculations. See (calc)GNU Emacs Calc Manual.

'`constants.el`' by Carsten Dominik ¶
Org can use names for constants in formulas in tables. Org can also use calculation suffixes for units, such as '`M`' for '`Mega`'. For a standard collection of such constants, install the '`constants`' package. Install version 2.0 of this package, available at https://github.com/cdominik/constants-for-Emacs. Org checks if the function `constants-get` has been autoloaded. Installation instructions are in the file '`constants.el`'.

'`cdlatex.el`' by Carsten Dominik ¶
Org mode can make use of the CDLaTeX package to efficiently enter LaTeX fragments into Org files. See Using CDLaTeX to enter math.

Imenu creates dynamic menus based on an index of items in a file. Org mode supports Imenu menus. Enable it with a mode hook as follows:

```
(add-hook 'org-mode-hook
          (lambda () (imenu-add-to-menubar "Imenu")))
```

By default the index is two levels deep—you can modify the depth using the option `org-imenu-depth`.

Org activates Imenu support only in the buffers opened after loading Imenu library. To enable Imenu support in an already opened Org buffer, reload Org.

'`speedbar.el`' by Eric M. Ludlam ¶

Speedbar package creates a special Emacs frame for displaying files and index items in files. Org mode supports Speedbar; users can drill into Org files directly from the Speedbar. The `<` in the Speedbar frame tweaks the agenda commands to that file or to a subtree.

‘`table.el`’ by Takaaki Ota ¶

Complex ASCII tables with automatic line wrapping, column- and row-spanning, and alignment can be created using the Emacs table package by Takaaki Ota. Org mode recognizes such tables and exports them properly. `C-c '` to edit these tables in a special buffer, much like Org’s code blocks. Because of interference with other Org mode functionality, Takaaki Ota tables cannot be edited directly in the Org buffer.

`C-c '` (`org-edit-special`) ¶

Edit a ‘`table.el`’ table. Works when point is in a ‘`table.el`’ table.

`C-c ~` (`org-table-create-with-table.el`) ¶

Insert a ‘`table.el`’ table. If there is already a table at point, this command converts it between the ‘`table.el`’ format and the Org mode format. See the documentation string of the command `org-convert-table` for the restrictions under which this is possible.

17.14.2 Packages that conflict with Org mode

In Emacs, shift-selection combines motions of point with shift key to enlarge regions. Emacs sets this mode by default. This conflicts with Org’s use of `S-<cursor>` commands to change timestamps, TODO keywords, priorities, and item bullet types, etc. Since `S-<cursor>` commands outside of specific contexts do not do anything, Org offers the variable `org-support-shift-select` for customization. Org mode accommodates shift selection by (i) making it available outside of the special contexts where special commands apply, and (ii) extending an existing active region even if point moves across a special context.

‘`cua.el`’ by Kim F. Storm ¶

Org key bindings conflict with `S-<cursor>` keys used by CUA mode. For Org to relinquish these bindings to CUA mode, configure the variable `org-replace-disputed-keys`. When set, Org moves the following key bindings in Org files, and in the agenda buffer—but not during date selection.

`S-UP` ⇒ `M-p`	`S-DOWN` ⇒ `M-n`
`S-LEFT` ⇒ `M--`	`S-RIGHT` ⇒ `M-+`
`C-S-LEFT` ⇒ `M-S--`	`C-S-RIGHT` ⇒ `M-S-+`

Yes, these are unfortunately more difficult to remember. If you want to have other replacement keys, look at the variable `org-disputed-keys`.

‘`ecomplete.el`’ by Lars Magne Ingebrigtsen ¶

Ecomplete provides “electric” address completion in address header lines in message buffers. Sadly Orgtbl mode cuts Ecomplete’s power supply: no completion happens when Orgtbl mode is enabled in message buffers while entering text in address header lines. If one wants to use

ecomplete one should *not* follow the advice to automagically turn on Orgtbl mode in message buffers (see The Orgtbl Minor Mode), but instead—after filling in the message headers—turn on Orgtbl mode manually when needed in the messages body.

‘`filladapt.el`’ by Kyle Jones ¶

Org mode tries to do the right thing when filling paragraphs, list items and other elements. Many users reported problems using both ‘`filladapt.el`’ and Org mode, so a safe thing to do is to disable filladapt like this:

```
(add-hook 'org-mode-hook 'turn-off-filladapt-mode)
```

‘`viper.el`’ by Michael Kifer ¶

Viper uses `C-c /` and therefore makes this key not access the corresponding Org mode command `org-sparse-tree`. You need to find another key for this command, or override the key in `viper-vi-global-user-map` with

```
(define-key viper-vi-global-user-map "C-c /" 'org-sparse-tree)
```

‘`windmove.el`’ by Hovav Shacham ¶

This package also uses the `S-<cursor>` keys, so everything written in the paragraph above about CUA mode also applies here. If you want to make the windmove function active in locations where Org mode does not have special functionality on `S-<cursor>`, add this to your configuration:

```
;; Make windmove work in Org mode:
(add-hook 'org-shiftup-final-hook 'windmove-up)
(add-hook 'org-shiftleft-final-hook 'windmove-left)
(add-hook 'org-shiftdown-final-hook 'windmove-down)
(add-hook 'org-shiftright-final-hook 'windmove-right)
```

‘`yasnippet.el`’ ¶

The way Org mode binds the `TAB` key (binding to `[tab]` instead of `"\t"`) overrules YASnippet’s access to this key. The following code fixed this problem:

```
(add-hook 'org-mode-hook
          (lambda ()
            (setq-local yas/trigger-key [tab])
            (define-key yas/keymap [tab] 'yas/next-field-or-maybe-expand)))
```

The latest version of YASnippet does not play well with Org mode. If the above code does not fix the conflict, start by defining the following function:

```
(defun yas/org-very-safe-expand ()
  (let ((yas/fallback-behavior 'return-nil)) (yas/expand)))
```

Then, tell Org mode to use that function:

```
(add-hook 'org-mode-hook
          (lambda ()
            (make-variable-buffer-local 'yas/trigger-key)
            (setq yas/trigger-key [tab])
            (add-to-list 'org-tab-first-hook 'yas/org-very-safe-expand)
            (define-key yas/keymap [tab] 'yas/next-field)))
```

17.15 Using Org on a TTY

Org provides alternative key bindings for TTY and modern mobile devices that cannot perform movement commands on point and key bindings with modifier keys. Some of these workarounds may be more cumbersome than necessary. Users should look into customizing these further based on their usage needs. For example, the normal `S-<cursor>` for editing timestamp might be better with `C-c .` chord.

Default	Alternative 1	Speed key	Alternative 2
`S-TAB`	`C-u TAB`	`C`	
`M-LEFT`	`C-c C-x l`	`l`	`Esc LEFT`
`M-S-LEFT`	`C-c C-x L`	`L`	
`M-RIGHT`	`C-c C-x r`	`r`	`Esc RIGHT`
`M-S-RIGHT`	`C-c C-x R`	`R`	
`M-UP`	`C-c C-x u`		`Esc UP`
`M-S-UP`	`C-c C-x U`	`U`	
`M-DOWN`	`C-c C-x d`		`Esc DOWN`
`M-S-DOWN`	`C-c C-x D`	`D`	
`S-RET`	`C-c C-x c`		
`M-RET`	`C-c C-x m`		`Esc RET`
`M-S-RET`	`C-c C-x M`		
`S-LEFT`	`C-c LEFT`		
`S-RIGHT`	`C-c RIGHT`		
`S-UP`	`C-c UP`		
`S-DOWN`	`C-c DOWN`		
`C-S-LEFT`	`C-c C-x LEFT`		
`C-S-RIGHT`	`C-c C-x RIGHT`		
`C-c C-,`	`C-c C-x s`		

17.16 Protocols for External Access

Org protocol is a tool to trigger custom actions in Emacs from external applications. Any application that supports calling external programs with an URL as argument may be used with this functionality. For example, you can configure bookmarks in your web browser to send a link to the current page to Org and create a note from it using capture (see Capture). You can also create a bookmark that tells Emacs to open the local source file of a remote website you are browsing.

In order to use Org protocol from an application, you need to register '`org-protocol://`' as a valid scheme-handler. External calls are passed to Emacs through the '`emacsclient`' command, so you also need to ensure an Emacs server is running. More precisely, when the application calls

```
emacsclient "org-protocol://PROTOCOL?key1=val1&key2=val2"
```

Emacs calls the handler associated to *PROTOCOL* with argument '`(:key1 val1 :key2 val2)`'.

Org protocol comes with three predefined protocols, detailed in the following sections. Configure `org-protocol-protocol-alist` to define your own.

17.16.1 The `store-link` protocol

Using the `store-link` handler, you can copy links, to that they can be inserted using `M-x org-insert-link` or yanking. More precisely, the command

```
emacsclient "org-protocol://store-link?url=URL&title=TITLE"
```

stores the following link:

```
[[URL][TITLE]]
```

In addition, *URL* is pushed on the kill-ring for yanking. You need to encode *URL* and *TITLE* if they contain slashes, and probably quote those for the shell.

To use this feature from a browser, add a bookmark with an arbitrary name, e.g., '`Org: store-link`' and enter this as *Location*:

```
javascript:location.href='org-protocol://store-link?' +
      new URLSearchParams({url:location.href, title:document.title});
```

Title is an optional parameter. Another expression was recommended earlier:

```
javascript:location.href='org-protocol://store-link?url='+
      encodeURIComponent(location.href);
```

The latter form is compatible with older Org versions from 9.0 to 9.4.

17.16.2 The `capture` protocol

Activating the "capture" handler pops up a '`Capture`' buffer in Emacs, using acapture template.

```
emacsclient "org-protocol://capture?template=X&url=URL&title=TITLE&body=BODY"
```

To use this feature, add a bookmark with an arbitrary name, e.g., '`Org: capture`', and enter this as '`Location`':

```
javascript:location.href='org-protocol://capture?' +
      new URLSearchParams({
            template: 'x', url: window.location.href,
            title: document.title, body: window.getSelection()});
```

You might have seen another expression:

```
javascript:location.href='org-protocol://capture?template=x'+
      '&url='+encodeURIComponent(window.location.href)+
      '&title='+encodeURIComponent(document.title)+
      '&body='+encodeURIComponent(window.getSelection());
```

It is a bit more cluttered than the former one, but it is compatible with previous Org versions 9.0-9.4. In these versions encoding of space as "+" character was not supported by URI decoder.

The capture template to be used can be specified in the bookmark (like 'X' above). If unspecified, the template key is set in the variable `org-protocol-default-template-key`. The following template placeholders are available:

```
%:link          The URL
%:description   The webpage title
%:annotation    Equivalent to [[%:link][%:description]]
%i              The selected text
```

17.16.3 The `open-source` protocol

The `open-source` handler is designed to help with editing local sources when reading a document. To that effect, you can use a bookmark with the following location:

```
javascript:location.href='org-protocol://open-source?&url='+
      encodeURIComponent(location.href)
```

The variable `org-protocol-project-alist` maps URLs to local file names, by stripping URL parameters from the end and replacing the `:base-url` with `:working-directory` and `:online-suffix` with `:working-suffix`. For example, assuming you own a local copy of 'https://orgmode.org/worg/' contents at '/home/user/worg', you can set `org-protocol-project-alist` to the following

```
(setq org-protocol-project-alist
      '(("Worg"
         :base-url "https://orgmode.org/worg/"
         :working-directory "/home/user/worg/"
         :online-suffix ".html"
         :working-suffix ".org")))
```

If you are now browsing 'https://orgmode.org/worg/org-contrib/org-protocol.html' and find a typo or have an idea about how to enhance the documentation, simply click the bookmark and start editing.

However, such mapping may not always yield the desired results. Suppose you maintain an online store located at 'https://example.com/'. The local sources reside in '/home/user/example/'. It is common practice to serve all products in such a store through one file and rewrite URLs that do not match an existing file on the server. That way, a request to 'https://example.com/print/posters.html' might be rewritten on the server to something like 'https://example.com/shop/products.php/posters.html.php'. The `open-source` handler probably cannot find a file named '/home/user/example/print/posters.html.php' and fails.

Such an entry in `org-protocol-project-alist` may hold an additional property `:rewrites`. This property is a list of cons cells, each of which maps a regular expression to a path relative to the `:working-directory`.

Now map the URL to the path '`/home/user/example/products.php`' by adding `:rewrites` rules like this:

```
(setq org-protocol-project-alist
      '(("example.com"
         :base-url "https://example.com/"
         :working-directory "/home/user/example/"
         :online-suffix ".php"
         :working-suffix ".php"
         :rewrites (("example.com/print/" . "products.php")
                    ("example.com/$" . "index.php")))))
```

Since '`example.com/$`' is used as a regular expression, it maps '`https://example.com/`', '`https://example.com`', '`https://www.example.com/`' and similar to '`/home/user/example/index.php`'.

The `:rewrites` rules are searched as a last resort if and only if no existing file name is matched.

Two functions can help you filling `org-protocol-project-alist` with valid contents: `org-protocol-create` and `org-protocol-create-for-org`. The latter is of use if you're editing an Org file that is part of a publishing project.

17.17 Org Crypt

Org Crypt encrypts the text of an entry, but not the headline, or properties. Behind the scene, it uses the (epa)Emacs EasyPG Library to encrypt and decrypt files, and EasyPG needs a correct (gnupg)GnuPG setup.

Any text below a headline that has a '`crypt`' tag is automatically encrypted when the file is saved. To use a different tag, customize the `org-crypt-tag-matcher` setting.

Here is a suggestion for Org Crypt settings in Emacs init file:

```
(require 'org-crypt)
(org-crypt-use-before-save-magic)
(setq org-tags-exclude-from-inheritance '("crypt"))

(setq org-crypt-key nil)
;; GPG key to use for encryption.
;; nil means  use symmetric encryption unconditionally.
;; "" means use symmetric encryption unless heading sets CRYPTKEY property.

(setq auto-save-default nil)
;; Auto-saving does not cooperate with org-crypt.el: so you need to
;; turn it off if you plan to use org-crypt.el quite often.  Otherwise,
;; you'll get an (annoying) message each time you start Org.

;; To turn it off only locally, you can insert this:
;;
;; # -*- buffer-auto-save-file-name: nil; -*-
```

It's possible to use different keys for different headings by specifying the respective key as property 'CRYPTKEY', e.g.:

```
* Totally secret :crypt:
  :PROPERTIES:
  :CRYPTKEY: 0x012345678901234567890123456789012345678 9
  :END:
```

Note that the 'CRYPTKEY' property is only effective when `org-crypt-key` is non-`nil`. If `org-crypt-key` is `nil`, Org uses symmetric encryption unconditionally.

Excluding the '`crypt`' tag from inheritance prevents already encrypted text from being encrypted again.

17.18 Org Mobile

Org Mobile is a protocol for synchronizing Org files between Emacs and other applications, e.g., on mobile devices. It enables offline-views and capture support for an Org mode system that is rooted on a "real" computer. The external application can also record changes to existing entries.

This appendix describes Org's support for agenda view formats compatible with Org Mobile. It also describes synchronizing changes, such as to notes, between the mobile application and the computer.

To change tags and TODO states in the mobile application, first customize the variables `org-todo-keywords`, `org-tag-alist` and `org-tag-persistent-alist`. These should cover all the important tags and TODO keywords, even if Org files use only some of them. Though the mobile application is expected to support in-buffer settings, it is required to understand TODO states *sets* (see Setting up keywords for individual files) and *mutually exclusive* tags (see Setting Tags) only for those set in these variables.

17.18.1 Setting up the staging area

The mobile application needs access to a file directory on a server[162] to interact with Emacs. Pass its location through the `org-mobile-directory` variable. If you can mount that directory locally just set the variable to point to that directory:

```
(setq org-mobile-directory "~/orgmobile/")
```

Alternatively, by using TRAMP (see (tramp)TRAMP User Manual), `org-mobile-directory` may point to a remote directory accessible through, for example, SSH, SCP, or DAVS:

```
(setq org-mobile-directory "/davs:user@remote.host:/org/webdav/")
```

With a public server, consider encrypting the files. Org also requires OpenSSL installed on the local computer. To turn on encryption, set the same password in the mobile application and in Emacs. Set the password in the variable `org-mobile-use-encryption`[163]. Note that even after the mobile application encrypts the file contents, the file name remains visible on the file systems of the local computer, the server, and the mobile device.

17.18.2 Pushing to the mobile application

The command `org-mobile-push` copies files listed in `org-mobile-files` into the staging area. Files include agenda files (as listed in `org-agenda-files`). Customize `org-mobile-files` to add other files. File names are staged with paths relative to `org-directory`, so all files should be inside this directory[164].

Push creates a special Org file '`agendas.org`' with custom agenda views defined by the user[165].

Finally, Org writes the file '`index.org`', containing links to other files. The mobile application reads this file first from the server to determine what other files to download for agendas. For faster downloads, it is expected to only read files whose checksums[166] have changed.

17.18.3 Pulling from the mobile application

The command `org-mobile-pull` synchronizes changes with the server. More specifically, it first pulls the Org files for viewing. It then appends captured entries and pointers to flagged or changed entries to the file '`mobileorg.org`' on the server. Org ultimately integrates its data in an inbox file format, through the following steps:

1. Org moves all entries found in '`mobileorg.org`'[167] and appends them to the file pointed to by the variable `org-mobile-inbox-for-pull`. It should reside neither in the staging area nor on the server. Each captured entry and each editing event is a top-level entry in the inbox file.
2. After moving the entries, Org processes changes to the shared files. Some of them are applied directly and without user interaction. Examples include changes to tags, TODO state, headline and body text. Entries requiring further action are tagged as '`FLAGGED`'. Org marks entries with problems with an error message in the inbox. They have to be resolved manually.
3. Org generates an agenda view for flagged entries for user intervention to clean up. For notes stored in flagged entries, Org displays them in the echo area when point is on the corresponding agenda item.
   ?
   Pressing `?` displays the entire flagged note in another window. Org also pushes it to the kill ring. To store flagged note as a normal note, use `? z C-y C-c C-c`. Pressing `?` twice does these things: first it removes the '`FLAGGED`' tag; second, it removes the flagged note from the property drawer; third, it signals that manual editing of the flagged entry is now finished.

From the agenda dispatcher, `?` returns to the view to finish processing flagged entries. Note that these entries may not be the most recent since the mobile application searches files that were last pulled. To get an updated agenda view with changes since the last pull, pull again.

17.19 Drag and Drop & `yank-media`

Org mode supports drag and drop (DnD) of files. By default, Org asks the user what must be done with the dropped file: attach it, insert '`file:`' link, or open the file. Customize `org-yank-dnd-method` to set the default DnD action.

When DnD method is “attach”, Org mode first consults DnD metadata to decide the attach method. For example, when file/files are dragged from a file manager, Org may attach by copying or by moving.

If Org cannot figure out which attachment method to use from the metadata, it defaults to `org-yank-dnd-default-attach-method` [168]

Starting from Emacs 29, Org mode supports `yank-media` command to yank images from the clipboard and files from a file manager.

When yanking images from clipboard, Org saves the image on disk and inserts the image link to Org buffer. Images are either saved as attachments to heading (default) or to a globally defined directory. The save location is controlled by `org-yank-image-save-method`.

The yanked images are saved under automatically generated name. You can customize `org-yank-image-file-name-function` to make Org query the image names or change the naming scheme.

When yanking files copied from a file manager, Org respects the value of `org-yank-dnd-method`. Image files pasted this way also respect the value of `org-yank-image-save-method` when the action to perform is ‘`attach`’.

Appendix A Hacking

This appendix describes some ways a user can extend the functionality of Org.

A.1 Hooks

Org has a large number of hook variables for adding functionality. A complete list of hooks with documentation is maintained by the Worg project at https://orgmode.org/worg/doc.html#hooks.

A.2 Add-on Packages

Various authors wrote a large number of add-on packages for Org. Some of these packages used to be part of the ‘`org-mode`’ repository but are now hosted in a separate ‘`org-contrib`’ repository here. A Worg page with more information is at: https://orgmode.org/worg/org-contrib/.

A.3 Adding Hyperlink Types

Org has many built-in hyperlink types (see Hyperlinks), and an interface for adding new link types. The following example shows the process of adding Org links to Unix man pages, which look like this

```
[[man:printf][The printf manual]]
```

The following ‘`ol-man.el`’ file implements it

```
;;; ol-man.el - Support for links to man pages in Org mode
(require 'ol)

(org-link-set-parameters "man"
                         :follow #'org-man-open
                         :export #'org-man-export
                         :store #'org-man-store-link)

(defcustom org-man-command 'man
  "The Emacs command to be used to display a man page."
  :group 'org-link
  :type '(choice (const man) (const woman)))

(defun org-man-open (path _)
  "Visit the manpage on PATH.
PATH should be a topic that can be thrown at the man command."
  (funcall org-man-command path))

(defun org-man-store-link (&optional _interactive?)
  "Store a link to a man page."
  (when (memq major-mode '(Man-mode woman-mode))
    ;; This is a man page, we do make this link.
    (let* ((page (org-man-get-page-name))
           (link (concat "man:" page))
           (description (format "Man page for %s" page)))
      (org-link-store-props
       :type "man"
       :link link
       :description description))))

(defun org-man-get-page-name ()
  "Extract the page name from the buffer name."
  ;; This works for both `Man-mode' and `woman-mode'.
  (if (string-match " \\(\\S-+\\)\\*" (buffer-name))
      (match-string 1 (buffer-name))
    (error "Cannot create link to this man page")))

(defun org-man-export (link description format _)
  "Export a man page link from Org files."
  (let ((path (format "http://man.he.net/?topic=%s&section=all" link))
        (desc (or description link)))
    (pcase format
      (`html (format "<a target=\"_blank\" href=\"%s\">%s</a>" path desc))
      (`latex (format "\\href{%s}{%s}" path desc))
      (`texinfo (format "@uref{%s,%s}" path desc))
      (`ascii (format "%s (%s)" desc path))
      (t path))))

(provide ol-man)
;;; ol-man.el ends here
```

To activate links to man pages in Org, enter this in the Emacs init file:

```
(require 'ol-man)
```

A review of ‘`ol-man.el`’:

1. First, ‘`(require 'ol)`’ ensures that ‘`ol.el`’ is loaded.

2. Then `org-link-set-parameters` defines a new link type with ‘`man`’ prefix and associates functions for following, exporting and storing such links. See the variable `org-link-parameters` for a complete list of possible associations.
3. The rest of the file implements necessary variables and functions.
   For example, `org-man-store-link` is responsible for storing a link when `org-store-link` (see Handling Links) is called from a buffer displaying a man page. It is passed an argument `interactive?` which this function does not use, but other store functions use to behave differently when a link is stored interactively by the user. It first checks if the major mode is appropriate. If check fails, the function returns `nil`, which means it isn’t responsible for creating a link to the current buffer. Otherwise the function makes a link string by combining the ‘`man:`’ prefix with the man topic. It also provides a default description. The function `org-insert-link` can insert it back into an Org buffer later on.

A.4 Adding Export Backends

Org’s export engine makes it easy for writing new backends. The framework on which the engine was built makes it easy to derive new backends from existing ones.

The two main entry points to the export engine are: `org-export-define-backend` and `org-export-define-derived-backend`. To grok these functions, see ‘`ox-latex.el`’ for an example of defining a new backend from scratch, and ‘`ox-beamer.el`’ for an example of deriving from an existing engine.

For creating a new backend from scratch, first set its name as a symbol in an alist consisting of elements and export functions. To make the backend visible to the export dispatcher, set `:menu-entry` keyword. For export options specific to this backend, set the `:options-alist`.

For creating a new backend from an existing one, set `:translate-alist` to an alist of export functions. This alist replaces the parent backend functions.

For complete documentation, see the Org Export Reference on Worg.

A.5 Tables in Arbitrary Syntax

Due to Org’s success in handling tables with Orgtbl, a frequently requested feature is the use of Org’s table functions in other modes, e.g., LaTeX. This would be hard to do in a general way without complicated customization nightmares. Moreover, that would take Org away from its simplicity roots that Orgtbl has proven. There is, however, an alternate approach to accomplishing the same.

This approach involves implementing a custom *translate* function that operates on a native Org *source table* to produce a table in another format. This strategy would keep the excellently working Orgtbl simple and isolate complications, if any, confined to the translate function. To add more alien table formats, we just add more translate functions. Also the burden of developing custom translate functions for new table formats is in the hands of those who know those formats best.

A.5.1 Radio tables

Radio tables are target locations for translated tables that are not near their source. Org finds the target location and inserts the translated table.

The key to finding the target location is the magic words ‘`BEGIN/END RECEIVE ORGTBL`’. They have to appear as comments in the current mode. If the mode is C, then:

```
/* BEGIN RECEIVE ORGTBL table_name */
/* END RECEIVE ORGTBL table_name */
```

At the location of source, Org needs a special line to direct Orgtbl to translate and to find the target for inserting the translated table. For example:

```
#+ORGTBL: SEND table_name translation_function arguments ...
```

‘`table_name`’ is the table’s reference name, which is also used in the receiver lines, and the ‘`translation_function`’ is the Lisp function that translates. This line, in addition, may also contain alternating key and value arguments at the end. The translation function gets these values as a property list. A few standard parameters are already recognized and acted upon before the translation function is called:

‘`:skip N`’
Skip the first N lines of the table. Hlines do count; include them if they are to be skipped.

‘`:skipcols (n1 n2 ...)`’
List of columns to be skipped. First Org automatically discards columns with calculation marks and then sends the table to the translator function, which then skips columns as specified in ‘`skipcols`’.

To keep the source table intact in the buffer without being disturbed when the source file is compiled or otherwise being worked on, use one of these strategies:

- Place the table in a block comment. For example, in C mode you could wrap the table between ‘`/*`’ and ‘`*/`’ lines.
- Put the table after an “end” statement. For example `\bye` in TeX and `\end{document}` in LaTeX.
- Comment and un-comment each line of the table during edits. The `M-x orgtbl-toggle-comment` command makes toggling easy.

A.5.2 A LaTeX example of radio tables

To wrap a source table in LaTeX, use the ‘`comment`’ environment provided by ‘`comment.sty`’[169]. To activate it, put `\usepackage{comment}` in the document header. Orgtbl mode inserts a radio table skeleton[170] with the command `M-x orgtbl-insert-radio-table`, which prompts for a table name. For example, if ‘`salesfigures`’ is the name, the template inserts:

```
% BEGIN RECEIVE ORGTBL salesfigures
% END RECEIVE ORGTBL salesfigures
\begin{comment}
#+ORGTBL: SEND salesfigures orgtbl-to-latex
| | |
\end{comment}
```

The line '`#+ORGTBL: SEND`' tells Orgtbl mode to use the function `orgtbl-to-latex` to convert the table to LaTeX format, then insert the table at the target (receive) location named '`salesfigures`'. Now the table is ready for data entry. It can even use spreadsheet features[171]:

```
% BEGIN RECEIVE ORGTBL salesfigures
% END RECEIVE ORGTBL salesfigures
\begin{comment}
#+ORGTBL: SEND salesfigures orgtbl-to-latex
| Month | Days | Nr sold | per day |
|-------+------+---------+---------|
| Jan   |   23 |      55 |     2.4 |
| Feb   |   21 |      16 |     0.8 |
| March |   22 |     278 |    12.6 |
#+TBLFM: $4=$3/$2;%.1f
% $ (optional extra dollar to keep Font Lock happy, see footnote)
\end{comment}
```

After editing, `C-c C-c` inserts the translated table at the target location, between the two marker lines.

For hand-made custom tables, note that the translator needs to skip the first two lines of the source table. Also the command has to *splice* out the target table without the header and footer.

```
\begin{tabular}{lrrr}
Month & \multicolumn{1}{c}{Days} & Nr.\ sold & per day\\
% BEGIN RECEIVE ORGTBL salesfigures
% END RECEIVE ORGTBL salesfigures
\end{tabular}
%
\begin{comment}
#+ORGTBL: SEND salesfigures orgtbl-to-latex :splice t :skip 2
| Month | Days | Nr sold | per day |
|-------+------+---------+---------|
| Jan   |   23 |      55 |     2.4 |
| Feb   |   21 |      16 |     0.8 |
| March |   22 |     278 |    12.6 |
#+TBLFM: $4=$3/$2;%.1f
\end{comment}
```

The LaTeX translator function `orgtbl-to-latex` is already part of Orgtbl mode and uses a '`tabular`' environment to typeset the table and marks horizontal lines with `\hline`. For additional parameters to control output, see Translator functions:

'`:splice BOOLEAN`'
When {{{var(BOOLEAN}}} is non-`nil`, return only table body lines; i.e., not wrapped in '`tabular`' environment. Default is `nil`.

'`:fmt FMT`'
Format string to warp each field. It should contain '`%s`' for the original field value. For example, to wrap each field value in dollar symbol, you could use '`:fmt "$%s$"`'. Format can also wrap a property list with column numbers and formats, for example '`:fmt (2 "$%s$" 4 "%s\\%%")`'. In place of a string, a function of one argument can be used; the function must return a formatted string.

‘:efmt EFMT’
Format numbers as exponentials. The spec should have ‘%s’ twice for inserting mantissa and exponent, for example ‘"%s\\times10^{%s}"’. This may also be a property list with column numbers and formats, for example ‘:efmt (2 "$%s\\times10^{%s}$" 4 "$%s\\cdot10^{%s}$")’. After *EFMT* has been applied to a value, *FMT*—see above—is also applied. Functions with two arguments can be supplied instead of strings. By default, no special formatting is applied.

A.5.3 Translator functions

Orgtbl mode has built-in translator functions: `orgtbl-to-csv` (comma-separated values), `orgtbl-to-tsv` (TAB-separated values), `orgtbl-to-latex`, `orgtbl-to-html`, `orgtbl-to-texinfo`, `orgtbl-to-unicode` and `orgtbl-to-orgtbl`. They use the generic translator, `orgtbl-to-generic`, which delegates translations to various export backends.

Properties passed to the function through the ‘ORGTBL SEND’ line take precedence over properties defined inside the function. For example, this overrides the default LaTeX line endings, `\\`, with `\\[2mm]`:

```
#+ORGTBL: SEND test orgtbl-to-latex :lend " \\\\[2mm]"
```

For a new language translator, define a converter function. It can be a generic function, such as shown in this example. It marks a beginning and ending of a table with ‘!BTBL!’ and ‘!ETBL!’; a beginning and ending of lines with ‘!BL!’ and ‘!EL!’; and uses a TAB for a field separator:

```
(defun orgtbl-to-language (table params)
  "Convert the orgtbl-mode TABLE to language."
  (orgtbl-to-generic
   table
   (org-combine-plists
    '(:tstart "!BTBL!" :tend "!ETBL!" :lstart "!BL!" :lend "!EL!" :sep "\t")
    params)))
```

The documentation for the `orgtbl-to-generic` function shows a complete list of parameters, each of which can be passed through to `orgtbl-to-latex`, `orgtbl-to-texinfo`, and any other function using that generic function.

For complicated translations the generic translator function could be replaced by a custom translator function. Such a custom function must take two arguments and return a single string containing the formatted table. The first argument is the table whose lines are a list of fields or the symbol `hline`. The second argument is the property list consisting of parameters specified in the ‘#+ORGTBL: SEND’ line. Please share your translator functions by posting them to the Org users mailing list, at mailto:emacs-orgmode@gnu.org.

A.6 Dynamic Blocks

Org supports *dynamic blocks* in Org documents. They are inserted with begin and end markers like any other code block, but the contents are updated automatically by a user function.

You can insert a dynamic block with `org-dynamic-block-insert-dblock`, which is bound to `C-c C-x x` by default. For example, `C-c C-x x c l o c k t a b l e RET` inserts a table that updates the work time (see Clocking Work Time).

Dynamic blocks can have names and function parameters. The syntax is similar to source code block specifications:

```
#+BEGIN: myblock :parameter1 value1 :parameter2 value2 ...
  ...
#+END:
```

These commands update dynamic blocks:

C-c C-x C-u (`org-dblock-update`) ¶
Update dynamic block at point.

C-u C-c C-x C-u ¶
Update all dynamic blocks in the current file.

Before updating a dynamic block, Org removes content between the ‘`BEGIN`’ and ‘`END`’ markers. Org then reads the parameters on the ‘`BEGIN`’ line for passing to the writer function as a plist. The previous content of the dynamic block becomes erased from the buffer and appended to the plist under `:content`.

The syntax for naming a writer function with a dynamic block labeled ‘`myblock`’ is: `org-dblock-write:myblock`.

The following is an example of a dynamic block and a block writer function that updates the time when the function was last run:

```
#+BEGIN: block-update-time :format "on %m/%d/%Y at %H:%M"
  ...
#+END:
```

The dynamic block’s writer function:

```
(defun org-dblock-write:block-update-time (params)
  (let ((fmt (or (plist-get params :format) "%d. %m. %Y")))
    (insert "Last block update at: "
            (format-time-string fmt))))
```

To keep dynamic blocks up-to-date in an Org file, use the function, `org-update-all-dblocks` in hook, such as `before-save-hook`. The `org-update-all-dblocks` function does not run if the file is not in Org mode.

Dynamic blocks, like any other block, can be narrowed with `org-narrow-to-block`.

Next: Speeding Up Your Agendas, Previous: Dynamic Blocks, Up: Hacking [Contents][Index]

A.7 Special Agenda Views

Org provides a special hook to further limit items in agenda views: `agenda`, `agenda*`[172], `todo`, `alltodo`, `tags`, `tags-todo`, `tags-tree`. Specify a custom function that tests inclusion of every matched item in the view. This function can also skip as much as is needed.

For a global condition applicable to agenda views, use the `org-agenda-skip-function-global` variable. Org uses a global condition with `org-agenda-skip-function` for custom searching.

This example defines a function for a custom view showing TODO items with ‘`waiting`’ status. Manually this is a multi-step search process, but with a custom view, this can be automated as follows:

The custom function searches the subtree for the ‘`waiting`’ tag and returns `nil` on match. Otherwise it gives the location from where the search continues.

```
(defun my-skip-unless-waiting ()
  "Skip trees that are not waiting"
  (let ((subtree-end (save-excursion (org-end-of-subtree t))))
    (if (re-search-forward ":waiting:" subtree-end t)
        nil          ; tag found, do not skip
      subtree-end))) ; tag not found, continue after end of subtree
```

To use this custom function in a custom agenda command:

```
(org-add-agenda-custom-command
 '("b" todo "PROJECT"
   ((org-agenda-skip-function 'my-skip-unless-waiting)
    (org-agenda-overriding-header "Projects waiting for something: "))))
```

Note that this also binds `org-agenda-overriding-header` to a more meaningful string suitable for the agenda view.

Search for entries with a limit set on levels for the custom search. This is a general approach to creating custom searches in Org. To include all levels, use ‘`LEVEL>0`’[173]. Then to selectively pick the matched entries, use `org-agenda-skip-function`, which also accepts Lisp forms, such as `org-agenda-skip-entry-if` and `org-agenda-skip-subtree-if`. For example:

‘`(org-agenda-skip-entry-if 'scheduled)`’
Skip current entry if it has been scheduled.

‘`(org-agenda-skip-entry-if 'notscheduled)`’
Skip current entry if it has not been scheduled.

‘`(org-agenda-skip-entry-if 'deadline)`’
Skip current entry if it has a deadline.

‘`(org-agenda-skip-entry-if 'scheduled 'deadline)`’
Skip current entry if it has a deadline, or if it is scheduled.

‘`(org-agenda-skip-entry-if 'todo '("TODO" "WAITING"))`’
Skip current entry if the TODO keyword is TODO or WAITING.

‘`(org-agenda-skip-entry-if 'todo 'done)`’
Skip current entry if the TODO keyword marks a DONE state.

‘`(org-agenda-skip-entry-if 'timestamp)`’
Skip current entry if it has any timestamp, may also be deadline or scheduled.

‘`(org-agenda-skip-entry-if 'regexp "regular expression")`’
Skip current entry if the regular expression matches in the entry.

‘(org-agenda-skip-entry-if 'notregexp "regular expression")’
Skip current entry unless the regular expression matches.

‘(org-agenda-skip-subtree-if 'regexp "regular expression")’
Same as above, but check and skip the entire subtree.

The following is an example of a search for ‘`waiting`’ without the special function:

```
(org-add-agenda-custom-command
 '("b" todo "PROJECT"
   ((org-agenda-skip-function '(org-agenda-skip-subtree-if
                                'regexp ":waiting:"))
    (org-agenda-overriding-header "Projects waiting for something: "))))
```

A.8 Speeding Up Your Agendas

Some agenda commands slow down when the Org files grow in size or number. Here are tips to speed up:

- Reduce the number of Org agenda files to avoid slowdowns due to hard drive accesses.
- Reduce the number of DONE and archived headlines so agenda operations that skip over these can finish faster.
- Do not dim blocked tasks:

  ```
  (setq org-agenda-dim-blocked-tasks nil)
  ```

- Stop preparing agenda buffers on startup:

  ```
  (setq org-agenda-inhibit-startup t)
  ```

- Disable tag inheritance for agendas:

  ```
  (setq org-agenda-use-tag-inheritance nil)
  ```

- Disable parsing of some properties:

  ```
  (setq org-agenda-ignore-properties '(stats))
  ```

  This will disable parsing and updating statistic cookies.

These options can be applied to selected agenda views. For more details about generation of agenda views, see the docstrings for the relevant variables, and this dedicated Worg page for agenda optimization.

A.9 Extracting Agenda Information

Org provides commands to access agendas through Emacs batch mode. Through this command-line interface, agendas are automated for further processing or printing.

`org-batch-agenda` creates an agenda view in ASCII and outputs to standard output. This command takes one string parameter. When string consists of a single character, Org uses it as a key to `org-agenda-custom-commands`. These are the same ones available through the agenda dispatcher (see The Agenda Dispatcher).

This example command line directly prints the TODO list to the printer:

```
emacs -batch -l ~/.emacs -eval '(org-batch-agenda "t")' | lpr
```

When the string parameter length is two or more characters, Org matches it with tags/TODO strings. For example, this example command line prints items tagged with ‘`shop`’, but excludes items tagged with ‘`NewYork`’:

```
emacs -batch -l ~/.emacs                                      \
      -eval '(org-batch-agenda "+shop-NewYork")' | lpr
```

An example showing on-the-fly parameter modifications:

```
emacs -batch -l ~/.emacs                                      \
   -eval '(org-batch-agenda "a"                               \
           org-agenda-span (quote month)                      \
           org-agenda-include-diary nil                       \
           org-agenda-files (quote ("~/org/project.org")))'   \
   | lpr
```

which produces an agenda for the next 30 days from just the ‘`~/org/projects.org`’ file.

For structured processing of agenda output, use `org-batch-agenda-csv` with the following fields:

category
The category of the item

head
The headline, without TODO keyword, TAGS and PRIORITY

type
The type of the agenda entry, can be

`todo`	selected in TODO match
`tagsmatch`	selected in tags match
`diary`	imported from diary
`deadline`	a deadline
`scheduled`	scheduled
`timestamp`	appointment, selected by timestamp
`closed`	entry was closed on date
`upcoming-deadline`	warning about nearing deadline
`past-scheduled`	forwarded scheduled item
`block`	entry has date block including date

todo
The TODO keyword, if any

tags
All tags including inherited ones, separated by colons

date
The relevant date, like '`2007-2-14`'

time
The time, like '`15:00-16:50`'

extra
String with extra planning info

priority-l
The priority letter if any was given

priority-n
The computed numerical priority

If the selection of the agenda item was based on a timestamp, including those items with '`DEADLINE`' and '`SCHEDULED`' keywords, then Org includes date and time in the output.

If the selection of the agenda item was based on a timestamp (or deadline/scheduled), then Org includes date and time in the output.

Here is an example of a post-processing script in Perl. It takes the CSV output from Emacs and prints with a checkbox:

```
#!/usr/bin/perl

# define the Emacs command to run
$cmd = "emacs -batch -l ~/.emacs -eval '(org-batch-agenda-csv \"t\")'";

# run it and capture the output
$agenda = qx{$cmd 2>/dev/null};

# loop over all lines
foreach $line (split(/\n/,$agenda)) {
    # get the individual values
    ($category,$head,$type,$todo,$tags,$date,$time,$extra,
     $priority_l,$priority_n) = split(/,/,$line);
    # process and print
    print "[ ] $head\n";
}
```

A.10 Using the Property API

Here is a description of the functions that can be used to work with properties.

Function: org-entry-properties *&optional pom which* ¶
Get all properties of the entry at point-or-marker *POM*. This includes the TODO keyword, the tags, time strings for deadline, scheduled, and clocking, and any additional properties defined in the entry. The return value is an alist. Keys may occur multiple times if the property key was used several times. *POM* may also be `nil`, in which case the current entry is used. If *WHICH* is `nil` or `all`, get all properties. If *WHICH* is `special` or `standard`, only get that subclass.

Function: org-entry-get *pom property &optional inherit* ¶
Get value of *PROPERTY* for entry at point-or-marker *POM*. By default, this only looks at properties defined locally in the entry. If *INHERIT* is non-`nil` and the entry does not have the property, then also check higher levels of the hierarchy. If *INHERIT* is the symbol `selective`, use inheritance if and only if the setting of `org-use-property-inheritance` selects *PROPERTY* for inheritance.

Function: org-entry-delete *pom property* ¶
Delete the property *PROPERTY* from entry at point-or-marker *POM*.

Function: org-entry-put *pom property value* ¶
Set *PROPERTY* to *VALUES* for entry at point-or-marker POM.

Function: org-buffer-property-keys *&optional include-specials* ¶
Get all property keys in the current buffer.

Function: org-insert-property-drawer ¶
Insert a property drawer for the current entry.

Function: org-entry-put-multivalued-property *pom property &rest values* ¶
Set *PROPERTY* at point-or-marker *POM* to *VALUES*. *VALUES* should be a list of strings. They are concatenated, with spaces as separators.

Function: org-entry-get-multivalued-property *pom property* ¶
Treat the value of the property *PROPERTY* as a whitespace-separated list of values and return the values as a list of strings.

Function: org-entry-add-to-multivalued-property *pom property value* ¶
Treat the value of the property *PROPERTY* as a whitespace-separated list of values and make sure that *VALUE* is in this list.

Function: org-entry-remove-from-multivalued-property *pom property value* ¶
Treat the value of the property *PROPERTY* as a whitespace-separated list of values and make sure that *VALUE* is *not* in this list.

Function: org-entry-member-in-multivalued-property *pom property value* ¶
Treat the value of the property *PROPERTY* as a whitespace-separated list of values and check if *VALUE* is in this list.

User Option: org-property-allowed-value-functions ¶
Hook for functions supplying allowed values for a specific property. The functions must take a single argument, the name of the property, and return a flat list of allowed values. If ‘`:ETC`’ is one of the values, use the values as completion help, but allow also other values to be entered. The functions must return `nil` if they are not responsible for this property.

A.11 Using the Mapping API

Org has sophisticated mapping capabilities to find all entries satisfying certain criteria. Internally, this functionality is used to produce agenda views, but there is also an API that can be used to execute arbitrary functions for each or selected entries. The main entry point for this API is:

Function: org-map-entries *func &optional match scope &rest skip* ¶
Call *FUNC* at each headline selected by *MATCH* in *SCOPE*.

FUNC is a function or a Lisp form. With point positioned at the beginning of the headline, call the function without arguments. Org returns a list of return values of calls to the function.

To avoid preserving point, Org wraps the call to *FUNC* in `save-excursion` form. After evaluation, Org moves point to the end of the line that was just processed. Search continues from that point forward. This may not always work as expected under some conditions, such as if the current subtree was removed by a previous archiving operation. In such rare circumstances, Org skips the next entry entirely when it should not. To stop Org from such skips, make *FUNC* set the variable `org-map-continue-from` to a specific buffer position.

MATCH is a tags/property/TODO match. Org iterates only matched headlines. Org iterates over all headlines when *MATCH* is `nil` or `t`.

SCOPE determines the scope of this command. It can be any of:

`nil`
The current buffer, respecting the restriction, if any.

`tree`
The subtree started with the entry at point.

`region`
The entries within the active region, if any.

`file`
The current buffer, without restriction.

`file-with-archives`
The current buffer, and any archives associated with it.

`agenda`
All agenda files.

`agenda-with-archives`
All agenda files with any archive files associated with them.

list of filenames
If this is a list, all files in the list are scanned.

The remaining arguments are treated as settings for the scanner's skipping facilities. Valid arguments are:

`archive`
Skip trees with the 'ARCHIVE' tag.

`comment`
Skip trees with the COMMENT keyword.

function or Lisp form ¶

Used as value for `org-agenda-skip-function`, so whenever the function returns `t`, *FUNC* is called for that entry and search continues from the point where the function leaves it.

The mapping routine can call any arbitrary function, even functions that change meta data or query the property API (see Using the Property API). Here are some handy functions:

Function: org-todo *&optional arg* ¶
Change the TODO state of the entry. See the docstring of the functions for the many possible values for the argument *ARG*.

Function: org-priority *&optional action* ¶
Change the priority of the entry. See the docstring of this function for the possible values for *ACTION*.

Function: org-toggle-tag *tag &optional onoff* ¶
Toggle the tag *TAG* in the current entry. Setting *ONOFF* to either `on` or `off` does not toggle tag, but ensure that it is either on or off.

Function: org-promote ¶
Promote the current entry.

Function: org-demote ¶
Demote the current entry.

This example turns all entries tagged with ‘`TOMORROW`’ into TODO entries with keyword ‘`UPCOMING`’. Org ignores entries in comment trees and archive trees.

```
(org-map-entries '(org-todo "UPCOMING")
                 "+TOMORROW" 'file 'archive 'comment)
```

The following example counts the number of entries with TODO keyword ‘`WAITING`’, in all agenda files.

```
(length (org-map-entries t "/+WAITING" 'agenda))
```

Appendix B History and Acknowledgments

B.1 From Carsten

Org was born in 2003, out of frustration over the user interface of the Emacs Outline mode. I was trying to organize my notes and projects, and using Emacs seemed to be the natural way to go. However, having to remember eleven different commands with two or three keys per command, only to hide and show parts of the outline tree, that seemed entirely unacceptable to me. Also, when using outlines to take notes, I constantly wanted to restructure the tree, organizing it parallel to my thoughts and plans. *Visibility cycling* and *structure editing* were originally implemented in the package ‘`outline-magic.el`’, but quickly moved to the more general ‘`org.el`’. As this environment became comfortable for project planning, the next step was adding *TODO entries*, basic *timestamps*, and *table support*. These areas highlighted the two main goals that Org still has today: to be a new, outline-based, plain text mode with innovative and intuitive editing features, and to incorporate project planning functionality directly into a notes file.